MDC 95P0058

Nonlinear Control of Missiles

Kevin A. Wise
Jack L. Sedwick
Rowena L. Eberhardt

McDonnell Douglas Corporation
St. Louis, Missouri 63166

September 1995

Contract F49620-92-C-0057

Air Force Office of Scientific Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL REPORT SEP. 1992 TO SEP. 1995

4. TITLE AND SUBTITLE

Nonlinear Control of Missiles

5. FUNDING NUMBERS

F49620-92-C-0057

6. AUTHOR(S)

Kevin A. Wise (PI)
Jack L. Sedwick
Rowena L. Eberhardt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

McDonnell Douglas Corporation
P. O. Box 516
St. Louis Missouri, 63166

8. PERFORMING ORGANIZATION REPORT NUMBER

MDC 95P0058

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Marc Jacobs
AFOSR/NM
Directorate of Mathematical and Information Sciences
Bolling AFB DC 20332-6448

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem of designing flight control system for high performance anti-air missiles is discussed in this report. The research addresses three areas of missile autopilot design. The first area investigates the application of nonlinear H-infinity optimal control for an integrated aerodynamic/thrust vector controlled missile. The second area investigates the design of autopilots for missiles using on-off reaction control valve actuators. A sliding mode control law is developed using H-infinity design methods. The third area focused on developing nonconservative robustness analysis algorithms for gain scheduled missile autopilots.

14. SUBJECT TERMS			15. NUMBER OF PAGES
	missile autopilots robustness analysis uncertainties	variable structure control sliding modes nonlinear H-infinity	188 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

(Blank Page)

PREFACE

The research described in this final report was performed at McDonnell Douglas Aerospace - East, Saint Louis, Missouri under Contract F49620-92-C-0057, entitled "Nonlinear Control of Missiles." The program was managed by Dr. Marc Jacobs of the Dynamics and Control Branch, Directorate of Mathematical and Computer Sciences, Air Force Office of Scientific Research, Bolling Air Force Base, DC.

McDonnell Douglas's program manager and principal investigator was Dr. Kevin Wise. The research described herein was performed by Dr. Kevin Wise, Dr. Jackson Sedwick, and Ms. Rowena Eberhardt of McDonnell Douglas, in consultation with Professors Christopher Byrnes, Alberto Isidori, and Heinz Schattler of Washington University, Saint Louis.

The research reported here was conducted during the period September 1992 through September 1995.

(Blank Page)

TABLE OF CONTENTS

LIST OF PAGES

Title Page

i-viii

1-180

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF TABLES

LIST OF ACRONYMS

Acronym	
AFOSR	Air Force Office of Scientific Research
AOA	Angle of Attack
ARE	Algebraic Riccati Equation
CIC	Close In Combat
CPU	Core Processing Unit
HJI	Hamilton-Jacobi-Isaacs
IATVC	Integrated Aerodynamic and Thrust Vector Control
KYP	Kalman-Yakubovitch-Popov
LTI	Linear Time Invariant
LQ	Linear Quadratic
LQR	Linear Quadratic Regulator
MDC	McDonnell Douglas Corporation
NAMP	New Aircraft and Missile Products Division
PDE	Partial Differential Equation
PSF	Unit - lb/ft^2
RCV	Reaction Control Valve (Reaction Control Jets)
TPBVP	Two Point Boundary Value Problem
TVC	Thrust Vector Control
UAV	Unmanned Air Vehicle
VSC	Variable Structure Control
2-D	Two Dimensional
3-DOF	Three Degree of Freedom
6-DOF	Six Degree of Freedom

1 Introduction

In aircraft close-in-combat scenarios a large off-boresight angle targeting capability, or the ability to engage targets in the rear hemisphere, is a significant advantage. Super agility in missiles refers to this capability. The research documented in this report is focused on developing this capability for air-to-air missiles. Following a successful missile launch and separation, low dynamic pressures can render aerodynamic controls ineffective in performing an agile turn. When the main propulsion system ignites vectoring the thrust can provide this capability. As the velocity increases, the aerodynamic surfaces become more effective, and may be blended to further enhance agility In order for the missile to possess super agility some form of alternate control is needed.

New Air Force interests in missile alternate controls (reaction jets, thrust vectoring) to augment, or possibly eliminate, aerodynamic control surfaces poses a considerable control system design challenge. Low cost reaction jets are constant thrust devices which result in bang-bang type controls. Thrust vector control actuation systems have hard angle and rate limits and actuator nonlinearities. Blending these alternate controls with aerodynamic control surfaces combine current linear autopilot design problems, typically solved using linear robust control methods, with nonlinear controls in which design methodologies do not exist. In addition to the nonlinear actuation systems, the high angle-of-attack missile dynamics and aerodynamics are very nonlinear.

The *Nonlinear Control of Missiles* research was focused on developing innovative flight control system design concepts, algorithms, and design tools to address agile missile technology needs. Specific progress was made in the following three areas:

- Missile flight control using nonlinear H_∞.
- Sliding mode design for reaction jet controlled missiles.
- Nonconservative mixed uncertainty analysis algorithms.

1.1 Research Objectives, Accomplishments, and Transitions

The *Nonlinear Control of Missiles* research objectives address several aspects of missile flight control system design. The agile missile flight control problem requires controlling the missile's flight at high angles of attack using actuators that are nonlinear. The research objectives in applying nonlinear H_∞ optimal control and sliding mode design address the nonlinear aspects of this problem. The third research objective in developing nonconservative robustness analysis

algorithms addresses the need to analyze flight control systems about an operating point. Using linearized models of the missile's dynamics, nonconservative analysis algorithms are required to determine the flight control system's sensitivity to uncertain dynamics and aerodynamics.

The following paragraphs briefly summarize the research objective, accomplishments made, and transitions of the technology in each area.

Missile Flight Control Using Nonlinear H_∞

The research objective was to apply recently developed nonlinear H_∞ optimal control to missile flight control problems. Accomplishments include developing algorithms to approximate the solution of the Hamilton-Jacobi-Isaacs partial differential equation, developing software for implementation of this approach to general nonlinear problems, and application of this approach to a six dimensional missile flight control problem. The software has been transitioned to MDC's Guidance and Control Technology IRAD for further application to missile guidance and control problems. One journal paper [1] and three conference papers [2-4] were published presenting theory and application of the approach.

The method of successive approximations was used to obtain an approximate solution to characteristic equations for the Hamilton-Jacobi-Isaacs partial differential equation. Software was developed and connected to MATRIXx to implement the approach on a nonlinear missile flight control problem. The missile software used only the 1st approximation to the integral equations. The approach was applied to an agile missile using aero/TVC control actuators performing an agility turn. Simulation results show excellent performance. A six dimensional state vector was used in modeling the missile's dynamics.

FORTRAN software was developed that applies the successive approximation solution procedure to the characteristic equations for general nonlinear problems. Documentation to support the software is contained in Appendix A, with the FORTRAN listings given in Appendix B. The software can be obtained from the authors electronically (by email) at wisek@mdcgwy.mdc.com.

Sliding mode design for reaction jet controlled missiles.

The research objective was to develop high performance autopilots for blending reaction jet thrusters with aerodynamic control surfaces for anti-air missiles. Since low cost reaction jets (reaction control valves) are on-off devices, a variable structure control approach was used. Under this objective, algorithms for designing H_∞ sliding modes were developed.

The necessary theory extensions were made to design sliding mode control laws using H_∞ design methods. This new design method combines results from singular perturbation theory and high gain feedback control theory. The H_∞ sliding mode design algorithms were applied to a missile autopilot design problem. The resulting control laws were used to blend aerodynamic controls with on-off reaction control valves (RCVs) to maneuver a missile. A comparison of the improved disturbance rejection capabilities of the H_∞ sliding mode design was made with a linear quadratic based design. A conference paper [5] was written presenting the theory extensions and design results.

This research was transitioned to MDC's Air Force Alternate Control Technology (ACT) program, Contract No. F08630-92-C-0010. The ACT program is evaluating alternate missile control effectors (reaction jet thrusters, thrust vectoring) to improve missile agility. MS Eberhardt, who supported this AFOSR research, was also responsible for designing missile autopilots (for on-off reaction jets) on the ACT program. Her dual role (AFOSR researcher/project engineer) led to an improved understanding of sliding mode controllers, and improved autopilot design software for the ACT program.

Nonconservative mixed uncertainty analysis algorithm.

The research objective was to develop a parameter space based nonconservative analysis capability that can compute robust stability bounds on simultaneous real and complex uncertainties. Algorithms were developed that combined simulated annealing with conjugate gradient optimization. Under MDC IRAD, these algorithms were coded in FORTRAN and incorporated into a toolset used by MDC's aircraft and missile projects.

Significant progress was been made in the development of a software analysis tool capable of analyzing mixed uncertainties. The FORTRAN program developed is referred to as ROBUSTC. Sample analysis problems that have 3 or fewer uncertainties have been analyzed using ROBUSTC. Results computed using ROBUSTC have matched analytical calculations for these test problems. Two conference papers [6,7] were written describing results in this area.

Further progress was made in extending the approach to accommodate a large number ($n = 30$) of uncertain parameters. The modification to the approach was simply a "vectorizing" of the variation polynomial algorithm, making the algorithms more computationally efficient. The method does have a drawback in that for the first pass through the problem the determinant of a matrix must be calculated 2^n times. This only has to be performed once, at the beginning. This

feature of the variation polynomial algorithm does make the analysis of high dimensional problems CPU intensive.

The robustness analysis tools and experience developed under this AFOSR program have been transitioned to MDC's 4th Generation Escape System program, Contract No. F33615-92-C-2290. This joint Air Force/Navy program will flight test a new ejection seat at Holloman AFB during 1996. The flight test program has 14 launches planned from a rocket sled with a F-16 forebody. These robustness analysis tools have been integral to the design and analysis of the seat's flight control system. These tools are used to assess the sensitivity of the ejection seat to uncertain mass parameters and uncertain aerodynamics. This is of critical concern in the ejection seat problem because the seat's flight control system must accommodate crew members ranging from the 95% male to the 5% female (this changes both the mass properties and aerodynamic characteristics).

This technology was further disseminated to Navy and Air Force laboratory experts (and MDC engineers) through a two day flight control design workshop held in St. Louis, November 4-5, 1993. This intensive workshop presented key aspects of MDC's optimal control design methodology and robustness analysis process and tools. This workshop received high praise from the participants and MDC management. The presenters (Kevin Wise, Rowena Eberhardt, Joe Brinker, Mike Sharp) received MDC's New Aircraft and Missile Product (NAMP) Division Leading Edge Award.

1.2 Organization of the Report

Section 2 presents nonlinear H_∞ optimal control theory and missile autopilot design results. The missile simulation results presented here use nonlinear aerodynamics and have not been published previously. (Previous papers published preliminary results based upon linear aerodynamics with nonlinear dynamics.) Software for the successive approximation solution approach and documentation are contained in the appendices. Each chapter contains at its end the references used in the chapter.

Section 3 details the development of H_∞ sliding mode controllers for missile autopilots. A complete derivation of the algorithms for developing a missile autopilot is presented. Simulation results for an agile missile maneuver are also presented.

Section 4 presents the research in developing nonconservative robustness analysis algorithms. The algorithms and application results are presented.

1.3 Chapter 1 References

[1]. K. Wise and J. Sedwick, "Nonlinear H_∞ Optimal Control For Agile Missiles," accepted for publication in the AIAA Journal of Guidance, Control, and Dynamics.

[2]. K. Wise and J. Sedwick, "Nonlinear H_∞ Optimal Control For Agile Missiles," Proc of the 1995 AIAA GNC Conference, Balt. MD, Aug. 1995, pp. 1295-1307.

[3]. K. Wise and J. Sedwick, "Successive Approximation Solution to the Hamilton-Jacobi-Isaacs Equation," Proc of the 33rd IEEE CDC, Orlando FL, Dec. 1994, pp. 1387-1391.

[4]. K. Wise and J. Sedwick, "Missile Autopilot Design Using Nonlinear H_∞ Optimal Control," Proc. of the 13th IFAC Symp. Automatic Control in Aerospace, Palo Alto CA, Sept. 1994, pp. 128-134.

[5]. R. Eberhardt and K. Wise, "High Gain H∞ Feedback Control for an Anti-Air Missile," Proc. of the 32nd IEEE CDC, San Antonio, TX, Dec. 13-17, 1993.

[6]. K. Wise, "A Parameter Space Robustness Test for Real and Complex Uncertainties," Proc. of the 32nd IEEE CDC, San Antonio, TX, Dec. 13-17, 1993.

[7]. K. Wise and K. Reddy, "Missile Autopilot Robustness To Real and Complex Uncertainties Using A Parameter Space Robustness Test," Proc. of the AIAA Guidance, Navigation, and Control Conference, Monterey, CA, August, 1993, Paper AIAA 93-3738, pp. 336-346.

2 Agile Missile Flight Control Using Nonlinear H_∞

Consider the nonlinear system modeled by the equations of the form

$$\begin{aligned} \dot{x} &= f(x) + g_1(x)u + g_2(x)w \\ z &= h(x) + d_1(x)u + d_2(x)w \end{aligned} \tag{2.1}$$

The first equation describes a plant with state x, defined on a neighborhood X of the origin in R^n with control input $u \in R^m$ and subject to a set of exogenous input variables $w \in R^r$ which includes commands to be tracked and/or disturbances to be rejected. The second equation defines the regulated variables $z \in R^s$ which may include tracking errors, for instance the difference between the actual plant output and its desired reference behavior, expressed as a function of some of the exogenous variables w, as well as a cost of the input u needed to achieve the prescribed control goal. The mappings $f(x)$, $g_i(x)$, $h(x)$, $d_i(x)$, in Eq. (2.1) are smooth mappings defined in a neighborhood of the origin in R^n. Also, it is assumed that $f(0) = 0$ and $h(0) = 0$.

The purpose of the control is two fold: to achieve closed loop stability and to attenuate the influence of the exogenous input w on the regulated variable z. A controller that locally asymptotically stabilizes the equilibrium $x = 0$ of the closed loop system is said to be an admissible controller. The requirement of disturbance attenuation may be dealt with in several different manners, depending on the specific class of exogenous signals to be considered and/or the performance criteria chosen to evaluate the regulated variables. The following characterization taken from [93] is considered here. Given a positive real number γ, it is said that the exogenous signals are locally attenuated by γ if there exists a neighborhood U of the point $x = 0$ such that for every $T > 0$ and for every piece wise continuous function $w{:}[0,T] \to R^r$ for which the initial state $x(0) = 0$ remains in U for all $t \in [0,T]$, the response $z{:}[0,T] \to R^s$ of Eq. (2.1) satisfies

$$\int_0^T z^T(\tau)z(\tau)d\tau \le \gamma^2 \int_0^T w^T(\tau)w(\tau)d\tau \tag{2.2}$$

The problem of local disturbance attenuation with internal stability is to find an admissible controller yielding local attenuation of the exogenous inputs.

The state space solution of linear H_∞ optimal control problems can be found in [94]. This same problem of reducing the H_∞ norm of a closed loop system has been viewed as a two person, zero sum, differential game in [95], where the solution is related to certain algebraic Riccati equations. This approach, for nonlinear systems has been pursued in [96] (also in [95]). For

nonlinear systems the Riccati equation is replaced with a particular Hamilton-Jacobi equation known as the Isaacs equations ([97], p. 67, Eq. (4.2.1)). The problem of disturbance attenuation in nonlinear systems requires the solution of the Hamilton-Jacobi-Isaacs (HJI) equation. van der Schaft [91] addresses this issue and shows that if the linearized system (at an equilibrium point) is such that the linear H_∞ optimal control exists, then the HJI equation is solvable (locally) and the corresponding state feedback solution has the desired stabilizing properties.

This problem of disturbance attenuation in nonlinear systems is applied here to an agile missile flight control problem. A state feedback control is constructed to control the missile in performing a high angle-of-attack maneuver in order to intercept a target in the rear hemisphere. The nonlinear H_∞ optimal control is found by solving the HJI equation (locally) by transforming the partial differential equation into a two point boundary value problem (TPBVP) using the method of characteristics, and approximating the integral solution of the TPBVP using successive approximations. The next section details this solution approach, followed by its application to the missile flight control problem. Appendix A of this report contains documentation for the FORTRAN software implementation of our solution approach.

2.1 Approximate Solution of the HJI Equation

Consider the problem of regulating the state x to x=0 by means of a state feedback control law $u = u(x)$. The design model is given by Eq. (2.1).

The HJI partial differential equation (PDE) can be expressed as

$$V_x^* f + h^T h - \begin{bmatrix} \frac{1}{2} g_1 V_x^{*T} + d_1 h \\ \frac{1}{2} g_2 V_x^{*T} + d_2 h \end{bmatrix}^T R^{-1} \begin{bmatrix} \frac{1}{2} g_1 V_x^{*T} + d_1 h \\ \frac{1}{2} g_2 V_x^{*T} + d_2 h \end{bmatrix} = 0 \qquad (2.3)$$

where the dependency on x has been dropped to shorten the expression, and

$$R = \begin{bmatrix} d_1^T(x) d_1(x) & d_1^T(x) d_2(x) \\ d_2^T(x) d_1(x) & d_2^T(x) d_2(x) - \gamma^2 I \end{bmatrix}$$

The solution approach used here forms the nonlinear H_∞ control law around a gain scheduled linear H_∞ solution, based upon the linearized dynamics, and then adds to the linear control law based upon an approximate solution to the HJI PDE Eq. (2.3) using the method of successive approximations. This represents a new approach to solving HJI PDE's.

Equation (2.3) can be written as

$$0 = h^T h - h^T S R^{-1} S^T h + V_x^* \left(f - B R^{-1} S^T h \right) - \tfrac{1}{4} V_x^* B R^{-1} B^T V_x^{*T} \tag{2.4}$$

where $B = [g_1(x) \quad g_2(x)]$, $S = [d_1(x) \quad d_2(x)]$, and γ in R results from a linear H_∞ design using the linearized dynamics about $x = 0$.

The state feedback control is given by

$$u = [1 \quad 0] \left(-R^{-1} \right) \left(\tfrac{1}{2} B^T V_x^{*T} + S^T h \right) \tag{2.5}$$

where the Lyapunov function $V^*(x)$ must satisfy the HJI PDE, Eq. (2.3).

An important special case of the nonlinear dynamics, Eq. (2.1), useful in aerospace applications is where the nonlinearities are confined only to $f(x)$. For this case

$$h(x) = Cx, \quad g_1(x) = G_1, \quad g_2(x) = G_2, \quad d_1(x) = D_1, \quad d_2(x) = D_2 \tag{2.6}$$

The nonlinearities in $f(x)$ are modeled as

$$f(x) = Ax + \Delta f(x) \tag{2.7}$$

where $\Delta f(x) = O(x^2)$.

The solution to the linear H_∞ problem is obtained from the following algebraic Riccati equation (ARE)

$$X\tilde{A} + \tilde{A}^T X + \tilde{Q} + X\tilde{R}X = 0 \tag{2.8}$$

where

$$B = [G_1 \quad G_2], \qquad S = [D_1 \quad D_2]$$

$$\tilde{Q} = C^T \left(I - S R^{-1} S^T \right) C, \quad \tilde{R} = -B R^{-1} B^T$$

$$\tilde{A} = A - B R^{-1} S^T C$$

The solution approach used here considers the nonlinearities $\Delta f(x)$ as generating a departure from the linearized H_∞ solution. Therefore, the Lyapunov function $V(x)$ is expressed as

$$V(x) = x^T X x + \Delta V(x) \tag{2.9}$$

where X is the solution to Eq. (2.8). $\Delta V(x)$ will be $O\left(x^4\right)$ and $x^T Xx$ will be the Lyapunov function for the linearized solution. Substituting Eq. (2.9) into the HJI PDE Eq. (2.4) and using Eqs. (2.6) through (2.8) the following first order PDE is obtained.

$$0 = 2x^T X\Delta f + \Delta V_x\left[\left(\tilde{A} + \tilde{R}X\right)x + \Delta f\right] + \frac{1}{4}\Delta V_x \tilde{R} \Delta V_x^T \tag{2.10}$$

Notice that $\Delta f(x) = 0$ implies that $\Delta V_x = 0$. Stability of the linearized solution implies that all of the eigenvalues of the $\tilde{A} + \tilde{R}X$ are in the left half plane.

Equation (2.10) is solved by the method of characteristics [Ford[9], pp. 230-231]. The formulas from Ford[9] apply to a PDE of first order in one dependent variable $(\xi = \Delta V)$ and n independent variables $(x_1 \quad \cdots \quad x_n)$. Let $p = \Delta V_x^T$, i.e.

$$p_1 = \frac{\partial \Delta V}{\partial x_1}, \quad \cdots, \quad p_n = \frac{\partial \Delta V}{\partial x_n} \tag{2.11}$$

The general first order PDE has the form

$$\theta(x_1, \cdots, x_n, p_1, \cdots, p_n, \xi) = 0 \tag{2.12}$$

The characteristic strips satisfy the following $2n+1$ differential equations

$$\begin{aligned} \frac{dx_i}{dt} &= \theta_{p_i} \\ \frac{dp_i}{dt} &= -\theta_{x_i} - \theta_\xi p_i \\ \frac{d\xi}{dt} &= \theta_{p_1} p_1 + \cdots + \theta_{p_n} p_n \end{aligned} \tag{2.13}$$

To further simplify notation define $\tilde{F} = \tilde{A} + \tilde{R}X$. Equation (2.12) becomes

$$\theta(x, p, \xi) = 2x^T X\Delta f + p^T\left[\tilde{F}x + \Delta f\right] + p^T \frac{\tilde{R}}{4} p = 0 \tag{2.14}$$

The characteristic equations for Eq. (2.14) are (with x replaced by z)

$$\frac{dz}{dt} = \left(\frac{\partial \theta(z, p, \xi)}{\partial p}\right)^T = \tilde{F}z + \frac{1}{2}\tilde{R}p + \Delta f(z) \tag{2.15}$$

$$\frac{dp}{dt} = -\left(\frac{\partial\theta(z,p,\xi)}{\partial z}\right)^T - \frac{\partial\theta(z,p,\xi)}{\partial\xi}p$$
$$= -\tilde{F}^T p - 2\Delta f_z^T Xz - \Delta f_z^T p - 2X\Delta f(z)$$

$$\frac{d\xi}{dt} = \left(\frac{\partial\theta(z,p,\xi)}{\partial p}\right)p = p^T\tilde{F}z + p^T\Delta f(z) - \frac{1}{2}p^T\tilde{R}p$$

where the independent variable t need not correspond to time. The last scalar equation in Eq. (2.15) need not be solved unless the term ΔV (Eq. (2.9)) is needed in computing the Lyapunov function.

Notice that an integral of Eq. (2.15) is

$$p^T\frac{\tilde{R}}{4}p + \left[\tilde{F}z + \Delta f(z)\right]^T p + 2z^T X\Delta f(z) = 0 \tag{2.16}$$

The characteristic equations Eq. (2.15) can be put into integral form in order to attempt a successive approximation solution procedure. The state and costate equations from Eq. (2.15) can be written as

$$\begin{bmatrix}\dot{z}(t)\\ \dot{p}(t)\end{bmatrix} = \begin{bmatrix}\tilde{F} & \frac{1}{2}\tilde{R}\\ 0 & -\tilde{F}^T\end{bmatrix}\begin{bmatrix}z(t)\\ p(t)\end{bmatrix} + \begin{bmatrix}\Delta f(z(t))\\ q(z(t),p(t))\end{bmatrix} \tag{2.17}$$

where

$$q(z,p) = -2\Delta f_z^T(z)Xz - \Delta f_z^T(z)p - 2X\Delta f(z) \tag{2.18}$$

and Δf is defined in Eq. (2.7). The integral of Eq. (2.17) is

$$\begin{bmatrix}z(t)\\ p(t)\end{bmatrix} = e^{\begin{bmatrix}\tilde{F} & \frac{1}{2}\tilde{R}\\ 0 & -\tilde{F}^T\end{bmatrix}t}\left\{\begin{bmatrix}z(0)\\ p(0)\end{bmatrix} + \int_0^t e^{-\begin{bmatrix}\tilde{F} & \frac{1}{2}\tilde{R}\\ 0 & -\tilde{F}^T\end{bmatrix}\tau}\begin{bmatrix}\Delta f(z)\\ q(z,p)\end{bmatrix}d\tau\right\} \tag{2.19}$$

Define

$$\Phi(t) = \begin{bmatrix}\phi_{11}(t) & \phi_{12}(t)\\ 0 & \phi_{22}(t)\end{bmatrix} = e^{\begin{bmatrix}\tilde{F} & \frac{1}{2}\tilde{R}\\ 0 & -\tilde{F}^T\end{bmatrix}t} \tag{2.20}$$

Differentiating the ϕ_{ij} yields

$$\begin{aligned}\dot{\phi}_{11}(t) &= \tilde{F}\phi_{11}(t)\\ \dot{\phi}_{12}(t) &= \tilde{F}\phi_{12}(t) + \tfrac{1}{2}\tilde{R}\phi_{22}(t)\\ \dot{\phi}_{22}(t) &= -\tilde{F}^T\phi_{22}(t)\end{aligned} \tag{2.21}$$

with initial conditions of $\phi_{11}(0) = I, \phi_{12}(0) = 0, \phi_{22}(0) = I$. Solving for the ϕ_{ij} yields

$$\begin{aligned}\phi_{11}(t) &= e^{\tilde{F}t}\\ \phi_{22}(t) &= e^{-\tilde{F}^T t}\\ \phi_{12}(t) &= e^{\tilde{F}t}\int_0^t e^{-\tilde{F}\tau}\tfrac{1}{2}\tilde{R}e^{-\tilde{F}^T\tau}d\tau\end{aligned} \tag{2.22}$$

Substituting these into Eq. (2.19) results in

$$z(t) = \phi_{11}(t)z_0 + \phi_{12}(t)p_0 + \int_0^t [\phi_{11}(t-\tau)\Delta f(\tau) + \phi_{12}(t-\tau)q(\tau)]d\tau \tag{2.23}$$

$$p(t) = \phi_{22}(t)p_0 + \int_0^t \phi_{22}(t-\tau)q(\tau)d\tau$$

where the dependency of $q(\bullet)$ and $\Delta f(\bullet)$ on z and p has been dropped for notational convenience. Solving for p_0 and substituting this into the $z(t)$ expression yields

$$z(t) = \phi_{11}(t)z_0 + \phi_{12}(t)\phi_{22}^{-1}(t)p(t) + \int_0^t \left[\phi_{11}(t-\tau)\Delta f(\tau) + \left\{\phi_{12}(t-\tau) - \phi_{12}(t)\phi_{22}^{-1}(t)\phi_{22}(t-\tau)\right\}q(\tau)\right]d\tau \int_0^t$$

(2.24)

Define

$$\begin{aligned}J(t) &= \phi_{12}(t)\phi_{22}^{-1}(t)\\ &= e^{\tilde{F}t}\int_0^t e^{-\tilde{F}\tau}\tfrac{1}{2}\tilde{R}e^{-\tilde{F}^T\tau}\,d\tau e^{\tilde{F}^T t} = \int_0^t e^{\tilde{F}(t-\tau)}\left(\tfrac{1}{2}\tilde{R}\right)e^{\tilde{F}^T(t-\tau)}d\tau\end{aligned} \tag{2.25}$$

Let $\zeta = t - \tau$. Then

$$J(t) = \int_0^t e^{\tilde{F}\zeta}\left(\tfrac{1}{2}\tilde{R}\right)e^{\tilde{F}^T\zeta}d\zeta \tag{2.26}$$

and

$$\begin{aligned} J &= e^{\tilde{F}\zeta}\,\tfrac{1}{2}\tilde{R}e^{\tilde{F}^T\zeta}\,\tilde{F}^{-T}\Big|_0^t - \tilde{F}\int_0^t e^{\tilde{F}\zeta}\,\tfrac{1}{2}\tilde{R}e^{\tilde{F}^T\zeta}\,d\zeta\tilde{F}^{-T} \\ &= e^{\tilde{F}t}\,\tfrac{1}{2}\tilde{R}e^{\tilde{F}^T t}\,\tilde{F}^{-T} - \tfrac{1}{2}\tilde{R}\tilde{F}^{-T} - \tilde{F}J\tilde{F}^{-T} \end{aligned} \tag{2.27}$$

Note that the integral $J(\infty)$ must exist since $\tilde{F}$ is a stable matrix. Several useful relationships from Eq. (2.27) are:

$$\frac{dJ}{dt} = e^{\tilde{F}t}\left(\tfrac{1}{2}\tilde{R}\right)e^{\tilde{F}^T t} \tag{2.28}$$

$$\frac{dJ}{dt} = \tilde{F}J + J\tilde{F}^T + \tfrac{1}{2}\tilde{R} \qquad J(0) = 0$$

$$\tilde{F}J + J\tilde{F}^T + \tfrac{1}{2}\tilde{R} + e^{\tilde{F}t}\left(-\tfrac{1}{2}\tilde{R}\right)e^{\tilde{F}^T t} = 0$$

$$\tilde{F}J(\infty) + J(\infty)\tilde{F}^T = -\tfrac{1}{2}\tilde{R}$$

In order to simplify the integral equation, Eq. (2.24), define the last term in the integral of Eq. (2.24) inside the $\{\ \}$ as

$$K(t,\tau) = \phi_{12}(t-\tau) - \phi_{12}(t)\phi_{22}^{-1}(\tau) \tag{2.29}$$

Using Eq. (2.21), and differentiating, we have

$$\begin{aligned} \frac{\partial K}{\partial t} &= \tilde{F}\phi_{12}(t-\tau) + \tfrac{1}{2}\tilde{R}\phi_{22}(t-\tau) - \left[\tilde{F}\phi_{12}(t) + \tfrac{1}{2}\tilde{R}\phi_{22}(t)\right]\phi_{22}^{-1}(\tau) \\ &= \tilde{F}\left[\phi_{12}(t-\tau) - \phi_{12}(t)\phi_{22}^{-1}(\tau)\right] + \tfrac{1}{2}\tilde{R}\left[\phi_{22}(t-\tau) - \phi_{22}(t)\phi_{22}^{-1}(\tau)\right] \\ &= \tilde{F}K(t,\tau) \end{aligned} \tag{2.30}$$

At $t = 0$ we have

$$
\begin{aligned}
K(0,\tau) &= \phi_{12}(-\tau) - \phi_{12}(0)\phi_{22}^{-1}(\tau) = \phi_{12}(-\tau) \\
&= e^{-\tilde{F}\tau}\int_0^{-\tau} e^{-\tilde{F}\zeta}\,\tfrac{1}{2}\tilde{R}e^{-\tilde{F}^T\zeta}\,d\zeta \\
&= -e^{-\tilde{F}\tau}\int_0^{\tau} e^{\tilde{F}\mu}\,\tfrac{1}{2}\tilde{R}e^{\tilde{F}^T\mu}\,d\mu \\
&= -e^{-\tilde{F}\tau}J(\tau)
\end{aligned}
\tag{2.31}
$$

Using Eq. (2.30) and (2.31), $K(t,\tau)$ can be expressed as

$$
\begin{aligned}
K(t,\tau) &= e^{\tilde{F}t}K(0,\tau) \\
&= -e^{\tilde{F}(t-\tau)}J(\tau) \\
&= -\phi_{11}(t-\tau)J(\tau)
\end{aligned}
\tag{2.32}
$$

By differentiating Eq. (2.32), Eq. (2.30) is obtained. Substituting these expressions into the $z(t)$ equation in Eq. (2.23) results in

$$
\begin{aligned}
z(t) &= \phi_{11}(t)z(0) + J(t)p(t) + \int_0^t \phi_{11}(t-\tau)\{\Delta f(\tau) - J(\tau)q(\tau)\}d\tau \\
&= e^{\tilde{F}t}z(0) + J(t)p(t) + e^{\tilde{F}t}\int_0^t e^{-\tilde{F}\tau}\{\Delta f(\tau) - J(\tau)q(\tau)\}d\tau
\end{aligned}
\tag{2.33}
$$

To solve for the costate $p(t)$ in Eq. (2.23) we have

$$
p(t) = e^{-\tilde{F}^T t}p(0) + e^{-\tilde{F}^T t}\int_0^t e^{\tilde{F}^T\tau}q(\tau)d\tau \tag{2.34}
$$

Multiplying by $e^{\tilde{F}^T t}$ results in

$$
e^{\tilde{F}^T t}p(t) = p(0) + \int_0^t e^{\tilde{F}^T\tau}q(\tau)d\tau \tag{2.35}
$$

The boundary condition for $p(t)$ at ∞ is known. Taking the limit as $t \to \infty$ yields

$$
0 \bullet p(\infty) = p(0) + \int_0^{\infty} e^{\tilde{F}^T\tau}q(\tau)d\tau \tag{2.36}
$$

Solving for $p(0)$, using the fact that $p(t) \to 0$ as $t \to \infty$, yields

$$p(0) = -\int_0^\infty e^{\tilde{F}^T \tau} q(\tau) d\tau \tag{2.37}$$

Substituting this into Eq. (2.34) yields the following expression for $p(t)$:

$$\begin{aligned} p(t) &= e^{-\tilde{F}^T t} p(0) + e^{-\tilde{F}^T t} \int_0^t e^{\tilde{F}^T \tau} q(\tau) d\tau \\ &= -e^{-\tilde{F}^T t} \left[\int_0^\infty e^{\tilde{F}^T \tau} q(\tau) d\tau - \int_0^t e^{\tilde{F}^T \tau} q(\tau) d\tau \right] \\ &= -e^{-\tilde{F}^T t} \int_t^\infty e^{\tilde{F}^T \tau} q(\tau) d\tau \end{aligned} \tag{2.38}$$

The resulting integral expressions for the characteristic equations in Eqs. (2.17) are

$$z(t) = e^{\tilde{F} t} z(0) + J(t) p(t) + e^{\tilde{F} t} \int_0^t e^{-\tilde{F} \tau} \left\{ \Delta f(z(\tau)) - J(\tau) q(z(\tau), p(\tau)) \right\} d\tau$$

$$p(t) = -e^{-\tilde{F}^T t} \int_t^\infty e^{\tilde{F}^T \tau} q(z(\tau), p(\tau)) d\tau \tag{2.39}$$

Solutions to these equations thus define a curve between $(x(0), p(0))$ $(x(0), p(0))$ and the equilibrium point (0,0). It is easy to verify (by differentiating) that these expressions satisfy Eqs. (2.17). Note that due to the stability of $\tilde{F}$, the integrals in Eq. (2.39) are defined for only mild restrictions on $\Delta f(x)$.

The successive approximation procedure used to solve the integral expressions in Eq. (2.39) begins with the solution of the linearized equations

$$\begin{aligned} z^{(0)}(t) &= e^{\tilde{F} t} z \\ p^{(0)}(t) &= 0 \end{aligned} \tag{2.40}$$

where the superscript on z and p denote the order of the approximation. These are then substituted into the right hand side of Eq. (2.39) to evaluate the next approximation, and so on.

After an acceptable degree of approximation has been achieved, the control is generated by evaluating $p(t)$ at $t = 0$. More precisely, the procedure is as follows.

To compute the state feedback control, Eq. (2.5), $\Delta V_x(x)$ is needed. To obtain $\Delta V_x(x)$ the following three step process is used.

1). Set the initial condition $z(0)$ in Eq. (2.39) to the current value of x .

2) Generate the desired degree of approximation, starting with Eq. (2.40).

3) Evaluate $p(0)$ and equate $\Delta V_x(x)$ to $p(0)$.

From Eq. (2.5) the control is then given by

$$u(x) = [1 \quad 0]\left(-R^{-1}\right)\left\{\tfrac{1}{2}B^T\left(2Xx + \Delta V_x^T\right) + S^T Cx\right\} \tag{2.41}$$

Local Contraction Mapping

In this section we prove that the successive approximation solution procedure provides a local contraction mapping. This is important in proving the existence and uniqueness of the solution.

Define

$$w(t) = \begin{bmatrix} z(t) \\ p(t) \end{bmatrix}$$

$$q[w(t)] = -2\Delta f_z^T[z(t)]Xz(t) - \Delta f_z^T[z(t)]p(t) - 2X\Delta f[z(t)]$$

$$J(t) = \int_0^t e^{\tilde{F}\tau}\tfrac{1}{2}\tilde{R}e^{\tilde{F}^T\tau}\,d\tau = e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T t} - \hat{\rho}$$

where $\hat{\rho}$ satisfies the Lyapunov equation $\tilde{F}\hat{\rho} + \hat{\rho}\tilde{F}^T = \tfrac{1}{2}\tilde{R}$. Let

$$s[w(t)] = \Delta f[z(t)] + \hat{\rho}q[w(t)]$$

Note that $q[w]$ and $s[w]$ are second order or higher in w since $\Delta f[w]$ is second order or higher.

The integral equations (2.39) can be rewritten using the above definitions as

$$z(t) = e^{\tilde{F}t}x - J(t)\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w(\tau)]d\tau + \int_0^t e^{\tilde{F}(t-\tau)}s[w(\tau)]d\tau - \int_0^t e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T\tau}q[w(\tau)]d\tau$$

$$p(t) = -\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w(\tau)]d\tau$$

Let Pw be the successive approximation mapping

$$(Pw)(t) = \begin{bmatrix} e^{\tilde{F}t}x - J(t)\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w(\tau)]d\tau + \int_0^t e^{\tilde{F}(t-\tau)}s[w(\tau)]d\tau - \int_0^t e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T\tau}q[w(\tau)]d\tau \\ -\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w(\tau)]d\tau \end{bmatrix}$$

It is shown that this is a contraction in a sufficiently small neighborhood of the equilibrium $[z=0, \quad p=0]$, as follows.

Let $\|\cdot\|_n$ denote the Euclidean norm in $\Re^n$. Let $\|\cdot\|_C$ be the norm on $C^{2n}[0,\infty)$, i.e. $\|w(\cdot)\|_C = \sup_{t\in[0,\infty)} \|w(\cdot)\|_{2n}$. The induced matrix norm of $e^{\tilde{F}t}$ is $\left\|e^{\tilde{F}t}\right\|_i = e^{\lambda t}$ where λ is the real part of the rightmost eigenvalue of $\tilde{F}$. Due to the stability of $\tilde{F}$, $\lambda < 0$. The induced matrix norm of $J(t)$ satisfies

$$\|J(t)\|_i \le \int_0^t \left\|e^{\tilde{F}\tau}\right\|_i \left\|\tfrac{1}{2}\tilde{R}\right\|_i \left\|e^{\tilde{F}^T\tau}\right\|_i d\tau = \int_0^t e^{2\lambda\tau}d\tau \left\|\tfrac{1}{2}\tilde{R}\right\|_i$$

$$= \left(\frac{1-e^{2\lambda t}}{-\lambda}\right)\left\|\tfrac{1}{2}\tilde{R}\right\|_i$$

Now, define

$$B_r = \left\{y \in \Re^{2n} : \|y\| \le r\right\}$$

$$w_0(t) = \begin{bmatrix} e^{\tilde{F}t}x \\ 0 \end{bmatrix}$$

$$S_r = \left\{w(\cdot) \in C^{2n}[0,\infty) : \|w(\cdot) - w_0(\cdot)\|_C \le r\right\}$$

Let $w_1(\cdot)$ and $w_2(\cdot)$ be arbitrary elements of S_r; then $w_1(t)$ and $w_2(t)$ lie in the ball B_r for all $t \in [0,\infty)$.

$$(Pw_1)(t)-(Pw_2)(t)=\begin{bmatrix} -J(t)\int_t^\infty e^{\tilde{F}^T(\tau-t)}\left(q[w_1(\tau)]-q[w_2(\tau)]\right)d\tau+\int_0^t e^{\tilde{F}(t-\tau)}\left(s[w_1(\tau)]-s[w_2(\tau)]\right)d\tau- \\ \int_0^t e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T\tau}\left(q[w_1(\tau)]-q[w_2(\tau)]\right)d\tau \\ -\int_t^\infty e^{\tilde{F}^T(\tau-t)}\left(q[w_1(\tau)]-q[w_2(\tau)]\right)d\tau \end{bmatrix}$$

Taking the norm on both sides yields

$$\begin{aligned} x\left\|(Pw_1)(t)-(Pw_2)(t)\right\| \le \|J(t)\|_i \int_t^\infty \left\|e^{\tilde{F}^T(\tau-t)}\right\|_i \kappa_q \|w_1(\tau)-w_2(\tau)\|_{2n}d\tau+\int_0^t \left\|e^{\tilde{F}(t-\tau)}\right\|_i \kappa_s \|w_1(\tau)-w_2(\tau)\|_{2n}d\tau \\ +\int_0^t \left\|e^{\tilde{F}t}\right\|_i \|\hat{\rho}\|_i \left\|e^{\tilde{F}^T\tau}\right\|_i \kappa_q \|w_1(\tau)-w_2(\tau)\|_{2n}d\tau+\int_t^\infty \left\|e^{\tilde{F}^T(\tau-t)}\right\|_i \kappa_q \|w_1(\tau)-w_2(\tau)\|_{2n}d\tau \end{aligned}$$

where κ_s and κ_q are Lipschitz constants in B_r for $s(\bullet)$ and $q(\bullet)$, respectively. These constants are finite due to the second order nature of $s(\bullet)$ and $q(\bullet)$. Evaluating the matrix norms and using *sup* norms on $w_1(t)-w_2(t)$ results in

$$\begin{aligned} \left\|(Pw_1)(t)-(Pw_2)(t)\right\| &\le \|J(t)\|_i \left\{\left[\frac{1-e^{2\lambda t}}{-\lambda}\frac{1-e^{\lambda t}}{-\lambda}\left(\frac{1}{2}\left\|\tilde{R}\right\|_i+\|\hat{\rho}\|_i\right)\frac{1-e^{\lambda t}}{-\lambda}\right]\kappa_q+\frac{1-e^{\lambda t}}{-\lambda}\kappa_s\right\}\|w_1(\bullet)-w_2(\bullet)\|_C \\ &\le \left[\left(\frac{1}{2\lambda^2}\left\|\tilde{R}\right\|_i+\frac{1}{-\lambda}\|\hat{\rho}\|_i\right)\kappa_q+\frac{1}{-\lambda}\kappa_s\right]\|w_1(\bullet)-w_2(\bullet)\|_C \end{aligned}$$

Since the expression on the right hand side does not involve t,

$$\left\|(Pw_1)(\bullet)-(Pw_2)(\bullet)\right\|_C \le \rho_r \|w_1(\bullet)-w_2(\bullet)\|_C$$

where

$$\rho_r=\left(\frac{1}{2\lambda^2}\left\|\tilde{R}\right\|_i+\frac{1}{-\lambda}\|\hat{\rho}\|_i\right)\kappa_q+\frac{1}{-\lambda}\kappa_s>0$$

The parameter ρ_r can be made less than one by choosing r small enough because of the 2nd order nature and Lipschitz continuity of $s(\bullet)$ and $q(\bullet)$. Therefore, P is a contraction mapping on S_r for $r>0$ sufficiently small.

To show that P maps S_r into itself for r sufficiently small, let $w(\bullet)\in S_r$. Then

$$(Pw)(t)-w_0(t)=\begin{bmatrix} -J(t)\int_t^\infty e^{\tilde{F}^T(\tau-t)}\left(q[w(\tau)]-q[w_0(\tau)]\right)d\tau+\int_0^t e^{\tilde{F}(t-\tau)}\left(s[w(\tau)]-s[w_0(\tau)]\right)d\tau- \\ \int_0^t e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T\tau}\left(q[w(\tau)]-q[w_0(\tau)]\right)d\tau \\ -\int_t^\infty e^{\tilde{F}^T(\tau-t)}\left(q[w(\tau)]-q[w_0(\tau)]\right)d\tau \end{bmatrix}$$

$$+\begin{bmatrix} -J(t)\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w_0(\tau)]d\tau+\int_0^t e^{\tilde{F}(t-\tau)}s[w_0(\tau)]d\tau-\int_0^t e^{\tilde{F}t}\hat{\rho}e^{\tilde{F}^T\tau}q[w_0(\tau)]d\tau \\ -\int_t^\infty e^{\tilde{F}^T(\tau-t)}q[w_0(\tau)]d\tau \end{bmatrix}$$

Taking norms results in

$$\|(Pw)(\bullet)-w_0(\bullet)\|_C\le\rho_r\|w(\bullet)-w_0(\bullet)\|_C+\frac{\|\tilde{R}\|}{2\lambda^2}\kappa_q\|x\|_n+\frac{1}{-\lambda}\kappa_s\|x\|_n+\|\hat{\rho}\|_i\frac{1}{-\lambda}\kappa_q\|x\|_n,$$

since

$$\|q[w_0(t)]\|_n\le\left\|q\left[e^{\tilde{F}t}x\right]\right\|_n\le\kappa_q\left\|e^{\tilde{F}t}x\right\|_n\le\kappa_q e^{\lambda t}\|x\|_n\le\kappa_q\|x\|_n$$

and similarly for $s[w_0(t)]$. This results in

$$\|(Pw)(\bullet)-w_0(\bullet)\|_C\le\rho_r\|w(\bullet)-w_0(\bullet)\|_C+\left(\frac{\|\tilde{R}\|}{2\lambda^2}\kappa_q+\frac{1}{-\lambda}\kappa_s+\|\hat{\rho}\|_i\frac{1}{-\lambda}\kappa_q\right)\|x\|_n$$

where the right hand side can be made as small as desired by choosing r small.

2.2 Missile Nonlinear H_∞ Optimal Control

This section presents the application of nonlinear H_∞ state feedback control to the agile missile flight control problem, and outlines the algorithms that are used in the successive approximation solution of the HJI equation. Only a pitch-plane autopilot is considered. The autopilot will command angle-of-attack (AOA) by thrust vectoring and deflecting the aerodynamic control surfaces.

The missile autopilot command (the control input) can be separated into linear and nonlinear components as

$$u = u_{LINEAR} + u_{NONLINEAR}$$

where u_{LINEAR} results from a gain scheduled linear H_∞ state feedback design, and $u_{NONLINEAR}$ is proportional to $\Delta V_x^T(x)$ which is obtained by solving the HJI PDE. The state feedback control is $u = -K(x)x$. The feedback gains $K(x)$ are calculated at a very fast sample rate (1000 Hz). Since this is proportional state feedback, the digital implementation has the identical form $u = -K(x)x$.

The remainder of this section describes the missile's pitch plane dynamics, the design model for the linear H_∞ state feedback design (u_{LINEAR}), and the calculation of $\Delta V_x^T(x)$ and $u_{NONLINEAR}$.

Missile Dynamics

The missile's rigid body pitch-plane short period dynamics are described by

$$\dot{\alpha} = \frac{1}{V_m}\left\{cos(\alpha)(G_z + Z_A + T_z) - sin(\alpha)(G_x + X_A + T_x)\right\} + q \qquad (2.42)$$

$$\dot{q} = M_A + M_T$$

where α is the angle-of-attack, q is the pitch rate, and

$$G_z = g\,cos(\theta); G_x = g\,sin(\theta)$$
$$Z_A = Z_\alpha \alpha + Z_\delta \delta_e$$
$$X_A = X_0 + X_\alpha \alpha + X_\delta \delta_e$$
$$M_A = M_0 + M_\alpha \alpha + M_\delta \delta_e + M_q q$$
$$T_z = -T/m\, sin(\delta_T); T_x = T/m\, cos(\delta_T)$$
$$M_T = -M_{TVC}\, sin(\delta_T)$$

where (G_x, G_z) models gravity, (X_A, Z_A) models normalized aerodynamic accelerations, M_A models normalized aerodynamic pitching moment, (T_x, T_z) models thrust forces normalized with respect to the mass, and M_T models the pitching moment produced by the thrust vectoring normalized by the pitch inertia. The variable δ_e models the pitch fin angle and δ_T models the pitch thrust vector angle.

In addition to the above dynamics the integrated aerodynamic/thrust vector control[10] (TVC) system has a common actuator that drives both δ_T and the aerodynamic control surfaces δ_e $(\delta_T = \mu\delta_e)$. These actuator dynamics are modeled using the following second order model:

$$\dddot{\delta} + 2\zeta\omega\ddot{\delta} + \omega^2\dot{\delta} = \omega^2\delta_c$$

Calculation of u_{LINEAR}

The autopilot design approach used here is similar to that for linear H_∞ control design[11]. The missile's dynamics are augmented with weighting filter states that will shape the sensitivity, complementary sensitivity, and control activity frequency responses. The design objectives are to shape the sensitivity $S(s)$ in order to follow AOA commands, to shape the complementary sensitivity $T(s)$ to roll off the plant, and to penalize the control activity $C(s)$.

The autopilot design requires the calculation of the "trim" state or equilibrium. This calculation was performed using the TRIM function from MATRIXx which operates directly on the nonlinear simulation. Trim is defined as $\dot{\alpha} = \dot{q} = \dot{\delta} = \ddot{\delta} = 0$. The output of the TRIM function was the steady state pitch rate q and actuator command δ_c for a specified AOA, Mach, altitude, and center of gravity.

The linearized missile dynamics about the trim point are modeled in the following state space model:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \underbrace{\begin{bmatrix} \bar{Z}_\alpha & 1 & \bar{Z}_\delta & 0 \\ M_\alpha & M_q & M_\delta & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\omega^2 & -2\zeta\omega \end{bmatrix}}_{A_p} \begin{bmatrix} \alpha \\ q \\ \delta \\ \dot{\delta} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ \omega^2 \end{bmatrix}}_{B_p} \delta_c$$

where

$$\bar{Z}_\alpha = \left.\frac{\partial\dot{\alpha}}{\partial\alpha}\right|_{TRIM}, \bar{Z}_\delta = \left.\frac{\partial\dot{\alpha}}{\partial\delta}\right|_{TRIM}$$

$$M_\alpha = \left.\frac{\partial\dot{q}}{\partial\alpha}\right|_{TRIM}, M_q = \left.\frac{\partial\dot{q}}{\partial q}\right|_{TRIM} M_\delta = \left.\frac{\partial\dot{q}}{\partial\delta}\right|_{TRIM}$$

The design objectives were realized by shaping the sensitivity $S(s)$ and complementary sensitivity $T(s)$ with weighting filters, and by penalizing the control activity $C(s) = W_C\ddot{\delta}$. The design of these weighting filters influences the bandwidth of the autopilot. To design these filters,

the filter coefficients were related to their 0 dB crossover frequency, denoted ω_c. As ω_c increases, the bandwidth increases as well as the amount of control activity.

The sensitivity weighting filter $W_S(s)$ is given by

$$W_S(s) = \frac{K(\tau s + 1)}{s}$$

The low frequency behavior of $W_S(s)$ demonstrates integrator characteristics in order to satisfy the AOA command tracking objective. The gain K was chosen such that the magnitude of $W_S(j\omega)$ was equal to -3 dB at ω_c. The zero was placed at ω_c.

The complementary sensitivity weighting filter $W_T(s)$ is given by

$$W_T(s) = \frac{K(\tau_n s + 1)}{(\tau_d s + 1)}$$

In designing $W_T(s)$ an ω_{min} was selected, $\tau_n = 1/\omega_{min}$ computed, and then the gain K was selected such that the magnitude of $W_T(j\omega)$ was 0 dB at ω_c. The denominator time constant was then arbitrarily selected to maintain a proper transfer function.

The weighting filters used in this study were:

$$W_S(s) = \frac{14.95(0.0473s + 1)}{s}$$

$$W_T(s) = \frac{0.444(0.0401s + 1)}{(0.001s + 1)}$$

$$W_C(s) = 0.01$$

These weighting filters were not scheduled with flight condition. State space models were created for $W_S - (A_S, B_S, C_S, D_S)$, and similarly, $W_T - (A_T, B_T, C_T, D_T)$, in order to connect the weighting filters with the missile dynamics.

The linear H_∞ design model is given by

$$\dot{x} = Ax + G_1 u + G_2 w$$
$$z = Cx + D_1 u + D_2 w$$

which is terms of the missile dynamics and weighting filters is

$$\begin{bmatrix} \dot{x} \\ \dot{x}_S \\ \dot{x}_T \end{bmatrix} = \underbrace{\begin{bmatrix} A_p & 0 & 0 \\ B_S C_p & A_S & 0 \\ B_T C_p & 0 & A_T \end{bmatrix}}_{A} \begin{bmatrix} x \\ x_S \\ x_T \end{bmatrix} + \underbrace{\begin{bmatrix} B_p \\ -B_S D_p \\ B_T D_p \end{bmatrix}}_{G_1} \delta_c + \underbrace{\begin{bmatrix} 0 \\ B_S \\ 0 \end{bmatrix}}_{G_2} \alpha_c$$

$$\begin{bmatrix} z_S \\ z_T \\ \dot{x}_C \end{bmatrix} = \underbrace{\begin{bmatrix} D_S C_p & C_S & 0 \\ D_T C_p & 0 & C_T \\ W_C C_c A_p & 0 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x \\ x_S \\ x_T \end{bmatrix} + \underbrace{\begin{bmatrix} D_S D_p \\ D_p \\ W_C C_c B_p \end{bmatrix}}_{D_1} \delta_c + \underbrace{\begin{bmatrix} -D_S \\ 0 \\ 0 \end{bmatrix}}_{D_2} \alpha_c$$

Using this model the ARE in Eq. (2.8) is solved to form the linear control at each design point in the flight envelope. The linear gain schedule was computed at the following design points:

AOA (deg)	Mach	Alt (Kft)	CG
-100	0.1	0	0
-90	0.6	10	1
-80	0.8	35	
-70	1.0		
-60	1.15		
-50	1.5		
-40	2.0		
-30	3.0		
-20	5.0		
-10			
-5			
0			
5			
10			
20			
30			
40			
50			
60			
70			
80			
90			
100			

At each of these design points a γ-iteration was performed to determine the level of disturbnace rejection. At each of these design points u_{LINEAR} is given by

$$\begin{aligned} u_{LINEAR} &= [1 \quad 0]\left(-R^{-1}\right)\left(B^T X + S^T C\right)x \\ &= -Kx \end{aligned}$$

Calculation of $u_{NONLINEAR}$

The state vector for this application is given by

$$x = \left[\alpha, q, \delta, \dot{\delta}, w_S, w_T\right]^T - x_{TRIM}^T$$

where the subscript "TRIM" refers to the equilibrium value, and the states w_S and w_T are from first order weighting filters used in the linear H_∞ design. This definition of the state vector satisfies the requirement that $f(0)=0$.

The elements of $f(x)$ in Eq. (2.1) are all linear except the first element, which is

$$f_1 = \frac{1}{V_m}\left[-\sin\alpha\left(X_0 + X_\alpha\alpha + X_{\delta_e}\delta_e + \frac{T}{m}\right) + \cos\alpha\left(Z_\alpha\alpha + Z_{\delta_e}\delta_e + \frac{T}{m}\mu\delta_e\right)\right] + q \quad (2.43)$$

where $\delta_T = \mu\delta_e$. Let $s\alpha_T$ and $c\alpha_T$ denote the $\sin\alpha_{TRIM}$ and $\cos\alpha_{TRIM}$, respectively. Expanding f_1 to third order terms results in

$$f_1 = \frac{1}{V_m}\left[-s\alpha_T X_T + c\alpha_T Z_T\right] + q_{TRIM} \quad (2.44)$$

$$+\frac{1}{V_m}\left[-s\alpha_T(X_\alpha + Z_T) - c\alpha_T(X_T + Z_\alpha)\right]x_1 + x_2 + \frac{1}{V_m}\left[-s\alpha_T X_{\delta_e} + c\alpha_T\left(Z_{\delta_e} + \frac{T}{m}\mu\right)\right]x_3$$

$$+\frac{1}{V_m}\left[-c\alpha_T\left(X_\alpha + \tfrac{1}{2}Z_T\right) + s\alpha_T\left(\tfrac{1}{2}X_T - Z_\alpha\right)X_T\right]x_1^2 + \frac{1}{V_m}\left[-c\alpha_T X_{\delta_e} - s\alpha_T\left(Z_{\delta_e} + \frac{T}{m}\mu\right)\right]x_1 x_3$$

$$+\frac{1}{V_m}\left[\tfrac{1}{2}s\alpha_T(X_\alpha + 3Z_T) + \tfrac{1}{6}c\alpha_T(X_T - 3Z_\alpha)\right]x_1^3 + \frac{1}{V_m}\left[\tfrac{1}{2}s\alpha_T X_{\delta_e} - \tfrac{1}{2}c\alpha_T\left(Z_{\delta_e} + \frac{T}{m}\mu\right)\right]x_1^2 x_3$$

where $X_T = X_0 + X_\alpha\alpha_T + X_{\delta_e}\delta_T + \frac{T}{m}$ and $Z_T = Z_\alpha\alpha_T + Z_{\delta_e}\delta_T + \frac{T}{m}\mu\delta_T$. The first line in this expansion adds to zero because of the definition of trim. The next three lines are linear in the states, and will be denoted as $A_{11}x_1$, $A_{12}x_2$, and $A_{13}x_3$, respectively. The remaining terms represent the higher order nonlinearities ($O(x^2)$) and are up to third order. These terms define $\Delta f_1(x)$. Rewriting these terms in a more compact form results in

$$\Delta f_1(x) = c_1 x_1^2 + c_2 x_1 x_3 + c_3 x_1^3 + c_4 x_1^2 x_3 \quad (2.45)$$

The vector $\Delta f(x)$ is given by

$$\Delta f(x) = \left[\Delta f_1(x), 0, 0, 0, 0, 0\right]^T \quad (2.46)$$

and

$$\Delta f_x^T(x) = \begin{bmatrix} 2c_1x_1 + c_2x_3 + 3c_3x_1^2 + 2c_4x_1x_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ c_2x_1 + c_4x_1^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.47}$$

The next step is to compute the nonlinear part of the control, $\Delta V_x^T(x)$. Only the first approximation is computed. Starting with the integral equations, Eq. (2.39) is

$$\Delta V_x^T(x) = -\int_0^\infty e^{\tilde{F}^T\tau}\left[-2X\Delta f(z(\tau)) - 2\Delta f_x^T(z(\tau))Xz(\tau)\right]d\tau \tag{2.48}$$

where the stable matrix $\tilde{F}$ and the positive definite symmetric matrix X are provided by an H_∞ design for the linear system, and $z(t)$ is the linear solution of the characteristic equations $z(t) = e^{\tilde{F}t}x$. In this application $\tilde{F}$ has distinct eigenvalues, so that if $\tilde{F}$ is diagonalized as $\tilde{F} = U\Lambda W^T$ then

$$e^{\tilde{F}\tau} = Ue^{\Lambda\tau}W^T = \sum_{i=1}^{n} u_i w_i^T e^{\lambda_i\tau} \tag{2.49}$$

$$e^{\tilde{F}^T\tau} = \sum_{i=1}^{n} w_i u_i^T e^{\lambda_i\tau}$$

where the u_i are right eigenvectors, w_i left eigenvectors, and $\Lambda = diag[\lambda_i]$ is the diagonal matrix of eigenvalues of $\tilde{F}$. Using Eq. (2.49), the state x is transformed using $y = W^T x$. This gives

$$z = Ue^{\Lambda\tau}y = \sum_{i=1}^{n} y_i e^{\lambda_i\tau}u_i \tag{2.50}$$

and

$$Xz = \sum_{i=1}^{n} y_i e^{\lambda_i\tau}Xu_i. \tag{2.51}$$

The nonlinear expressions for $\Delta f(z)$ and $\Delta f_x^T(z)$ can be written using Eqs. (2.50) and (2.51) as the following linear sums of the eigenvalues:

$$
\begin{aligned}
z_1 &= \sum_{j=1}^{n} y_j e^{\lambda_j \tau} u_{1j} \\
z_3 &= \sum_{j=1}^{n} y_j e^{\lambda_j \tau} u_{3j} \\
z_1^2 &= \sum_{j=1}^{n}\sum_{k=1}^{n} y_j y_k e^{(\lambda_j+\lambda_k)\tau} u_{1j} u_{1k} \\
z_1 z_3 &= \sum_{j=1}^{n}\sum_{k=1}^{n} y_j y_k e^{(\lambda_j+\lambda_k)\tau} u_{1j} u_{3k} \\
z_1^3 &= \sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} y_j y_k y_l e^{(\lambda_j+\lambda_k+\lambda_l)\tau} u_{1j} u_{1k} u_{1l} \\
z_1^2 z_3 &= \sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} y_j y_k y_l e^{(\lambda_j+\lambda_k+\lambda_l)\tau} u_{1j} u_{1k} u_{3l}
\end{aligned}
\tag{2.52}
$$

to obtain

$$
\begin{aligned}
\Delta f &= e_1 g^T z e_1^T z + e_1 h^T z e_1^T z e_1^T z \\
\Delta f_z^T &= \left(e_1 g^T + g e_1^T\right) z e_1^T + \left(2 e_1 h^T + h e_1^T\right) z e_1^T z e_1^T
\end{aligned}
\tag{2.53}
$$

where

$$
\begin{aligned}
e_1^T &= (1,0,0,0,0,0) \\
g^T &= (c_1,0,c_2,0,0,0) \\
h^T &= (c_3,0,c_4,0,0,0)
\end{aligned}
\tag{2.54}
$$

Inserting these and Eq. (2.49) into Eq. (2.48) results in

$$
\begin{aligned}
\Delta V_x^T(x) = W \int_0^{\infty} \Big[& e^{\Lambda\tau} 2U\left(e_1 g^T + g e_1^T\right) e^{\Lambda\tau} (y e_1 X U) e^{\Lambda\tau} + e^{\Lambda\tau}\left(2U^T X e_1 g^T U\right) e^{\Lambda\tau}\left(y e_1^T U\right) e^{\Lambda\tau} + \\
& e^{\Lambda\tau} 2U^T \left(2 e_1 h^T + h e_1^T\right) U e^{\Lambda\tau}\left(y e_1^T X U\right) e^{\Lambda\tau}\left(y e_1^T U\right) e^{\Lambda\tau} + e^{\Lambda\tau}\left(2U^T X e_1 h^T U\right) e^{\Lambda\tau}\left(y e_1^T U\right) e^{\Lambda\tau}\left(y e_1^T U\right) e^{\Lambda\tau} \Big] d\tau y
\end{aligned}
\tag{2.55}
$$

This can be integrated using the identities

$$\int_a^b \left[e^{\Lambda\tau} P e^{\Lambda\tau} Q e^{\Lambda\tau}\right]_{ij} d\tau = \sum_k \frac{P_{ik}Q_{kj}}{\lambda_i+\lambda_j+\lambda_k}\left(e^{(\lambda_i+\lambda_j+\lambda_k)b} - e^{(\lambda_i+\lambda_j+\lambda_k)a}\right) \tag{2.56}$$

$$\int_a^b \left[e^{\Lambda\tau} P e^{\Lambda\tau} Q e^{\Lambda\tau} R\right]_{ij} d\tau = \sum_k \sum_l \frac{P_{ik}Q_{kl}R_{lj}}{\lambda_i+\lambda_j+\lambda_k+\lambda_l}\left(e^{(\lambda_i+\lambda_j+\lambda_k+\lambda_l)b} - e^{(\lambda_i+\lambda_j+\lambda_k+\lambda_l)a}\right)$$

to obtain

$$\Delta V_x^T(x) = WMy \tag{2.57}$$

where

$$M_{ij} = -\sum_k S_{ijk}^{(3)} y_k - \sum_k \sum_l S_{ijkl}^{(4)} y_k y_l \tag{2.58}$$

with

$$S_{ijk}^{(3)} = \frac{\left[2\left(U^T e_1\right)_i \left(U^T g\right)_k + 2\left(U^T g\right)_i \left(U^T e_1\right)_k\right]\left(U^T X e_1\right)_j + 2\left(U^T X e_1\right)_i \left(U^T g\right)_k \left(U^T e_1\right)_j}{\lambda_i+\lambda_j+\lambda_k}$$

$$S_{ijkl}^{(4)} = \frac{[\eta]\left(U^T X e_1\right)_l \left(U^T e_1\right)_j + 2\left(U^T X e_1\right)_i \left(U^T h\right)_k \left(U^T e_1\right)_l \left(U^T e_1\right)_j}{\lambda_i+\lambda_j+\lambda_k+\lambda_l} \tag{2.59}$$

where

$$\eta = 4\left(U^T e_1\right)_i \left(U^T h\right)_k + 2\left(U^T h\right)_i \left(U^T e_1\right)_k. \tag{2.60}$$

Defining $T^{(3)}$ and $T^{(4)}$ arrays by

$$\begin{aligned} T_{jk}^{(3)} &= \sum_i w_i S_{ijk}^{(3)} \\ T_{jkl}^{(4)} &= \sum_i w_i S_{ijkl}^{(4)} \end{aligned} \tag{2.61}$$

results in a polynomial expression in the transformed states y. The gradient of the Lyapunov function $\Delta V_x^T(x)$ becomes

$$\Delta V_x^T(x) = \sum_j \sum_k T_{jk}^{(3)} y_j y_k + \sum_j \sum_k \sum_l T_{jkl}^{(4)} y_j y_k y_l \tag{2.62}$$

The coefficients $T_{jk}^{(3)}$ form a $6\times6\times6$ array and $T_{jkl}^{(4)}$ a $6\times6\times6\times6$ array, and can be calculated off-line. The nonlinear contribution to the control $u_{NONLINEAR}$ is given by

$$u_{NONLINEAR} = [1 \quad 0]\left(-R^{-1}\right)\tfrac{1}{2}B^T \Delta V_x^T(x)$$

2.3 Simulation Results

This section presents simulation results using the nonlinear H_∞ control law for agile missile flight control. In forming the nonlinear H_∞ control law several assumptions were introduced in modeling the missile nonlinearities. It is important to present these modeling assumptions so that it is clear what nonlinearities are present in this problem.

In the earlier versions of this research [Ref. 4 in Chapter 1], we modeled the missile's aerodynamics as a linear function that was constant with angle of attack. In the results presented here this assumption has been removed, and the missile's aerodynamics are nonlinear.

The TVC deflection $sin(\delta_{TVC})$ in the $\dot{\alpha}$ equation has been replaced by $\mu\delta_e$. This linearity assumption for the actuator deflection is appropriate here because the TVC actuator is mechanically limited to 10 degrees deflection. The actual TVC flow angle does not deflect the same amount as the nozzle angle due to losses in the nozzle. The losses further limit the actual vector angle, reducing it to approximately 8.5 degrees (in this application). Over this limited range the $sin(\delta_{TVC})$ is linear.

The nonlinearity f_1 described in Eq. (2.43) is modeled using a third order polynomial. This additional modeling assumption adequately captures the nonlinearities introduced by the $sin(\alpha)$ and $cos(\alpha)$. Modeling the nonlinearities using polynomials is an important aspect of our solution approach. The polynomial models make it significantly easier to solve the integral expressions for the characteristic equations.

The aerodynamic data used in this study was obtained from high angle-of-attack wind tunnel measurements in the McDonnell Douglas Polysonic Wind Tunnel in St. Louis using a 1/4 scale model. Figure 2.1 illustrates the wind tunnel test hardware.

Installation Drawings

- **MDC Has Built A New High AOA Support Mechanism**
- **AOA Up To 90° In The MDA Polysonic Wind Tunnel (PSWT)**
- **Includes Roll Control Pod**

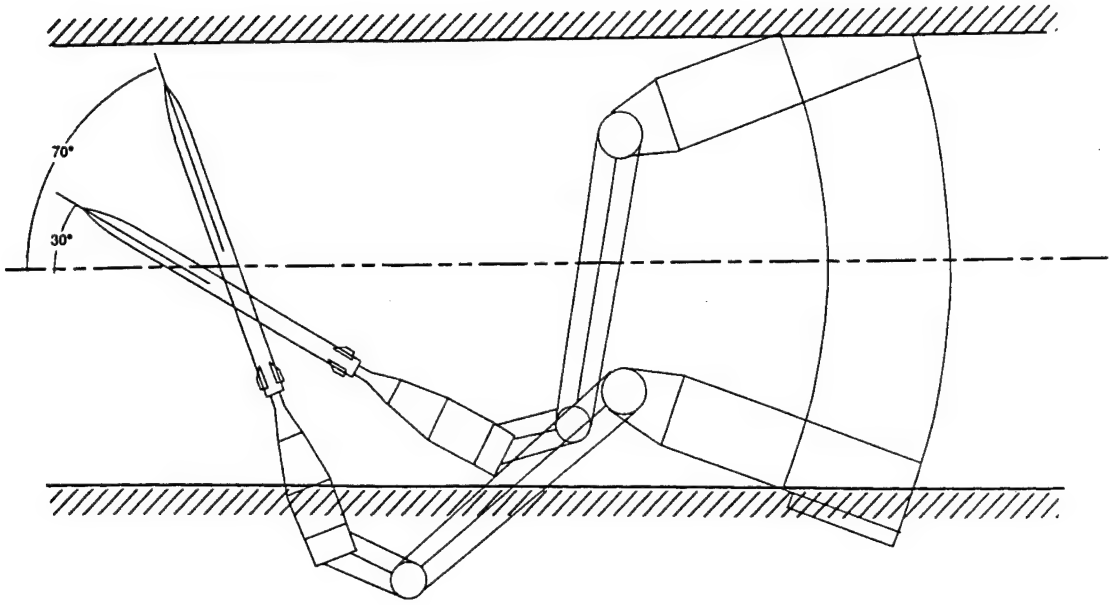


Figure 2.1 Wind tunnel test hardware.

Figure 2.2 shows the pitching moment coefficient as a function of angle of attack at Mach numbers of 0.45, 0.85, and 1.2. As the velocity of the missile changes the aerodynamics characteristics also change. Note that at Mach 0.45 the missile becomes unstable at approximately 30° AOA. As the velocity increases this transition region moves to lower AOA's.

Figure 2.3 illustrates the effect of aerodynamic fin deflections on the pitching moment. At low AOA's the fins provide adequate control power. As AOA increases, the aerodynamic controls are no longer effective. Note that there are sign reversals in the fin stability derivatives at the higher AOA's. By using thrust vectoring the problem of sign reversals at high AOAs is eliminated.

The missile autopilot commands an actuator that deflects the TVC nozzle. For the IATVC [10] actuator design used in this study (common actuator drives both the TVC nozzle and aerodynamic fins), the aerodynamic fins deflect 3 times the amount of the TVC nozzle.

The actuator command contains a linear part, from the gain scheduled linear H_∞ design, and a nonlinear part that is proportional to $\Delta V_x^T(x)$. Only the first approximation is computed in the calculation of $\Delta V_x^T(x)$. Computing the first approximation requires solving the integral in Eq. (2.48), where the stable matrix $\tilde{F}$ and the positive definite symmetric matrix X are provided by the gain scheduled linear H_∞ design (for the linearized system), and $z(t)$ is the linear solution of the characteristic equations $z(t) = e^{\tilde{F}t}x$.

In this application $\tilde{F}$ has distinct eigenvalues, so that if $\tilde{F}$ is diagonalized as $\tilde{F} = U\Lambda W^T$, then the integral expression in Eq. (2.48) can be solved by representing the matrix exponential with its modal expansion. This makes this integral expression a combination of polynomials weighted by exponentials, which are easy to integrate.

The gradient of the Lyapunov function $\Delta V_x^T(x)$ is solved for using Eq. (2.62). Figure 2.4 illustrates the calculations used in computing $\Delta V_x^T(x)$. FORTRAN subroutines were developed to implement these algorithms. (Documentation for this code is contained in Appendix A, with the software given in Appendix B.) Note that a linear gain schedule is calculated off-line and stored for use in computing the linear part of the control law.

Figure 2.5 shows our MATRIXx implementation of the nonlinear H_∞ control law. The super block has 8 inputs (α, altitude, Mach number, α command, pitch rate, TVC nozzle deflection, TVC nozzle rate, and fuel ratio (err: 1=full, 0=empty)). The fuel ratio is used to schedule the gains with changes in the center of gravity. The control is the TVC nozzle deflection command computed by the autopilot. This command has 3 terms:The trim deflection; the linear H_∞

Subsonic/Transonic Stability Trends

• Launch CG

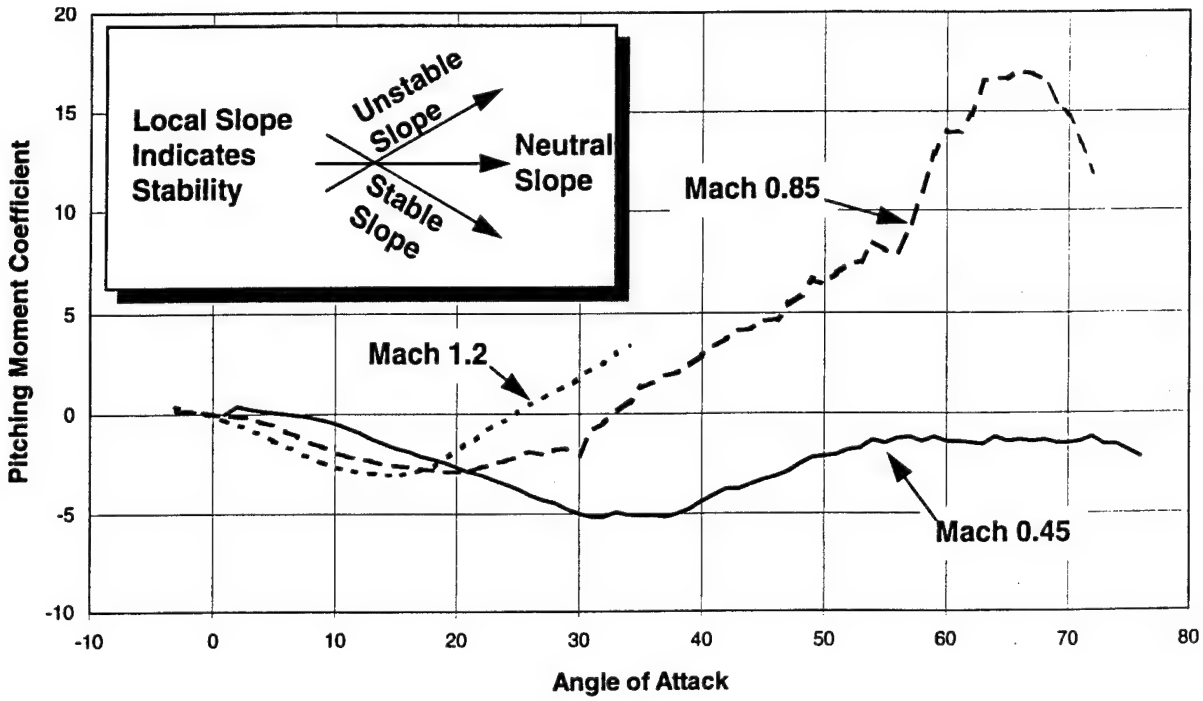


Figure 2.2 Pitching moment coefficient as a function of angle of attack.

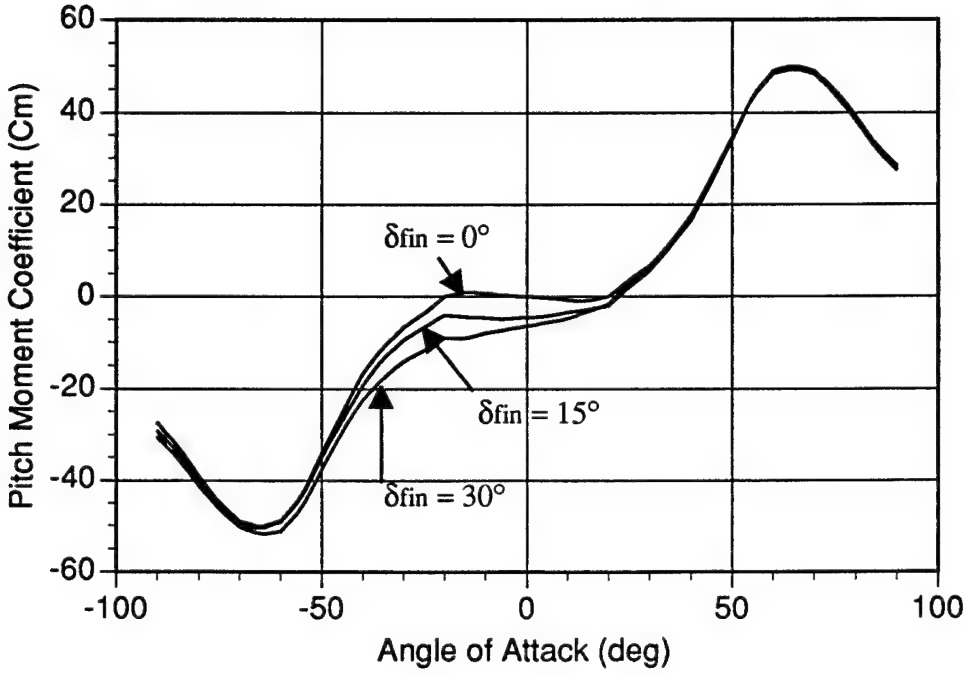
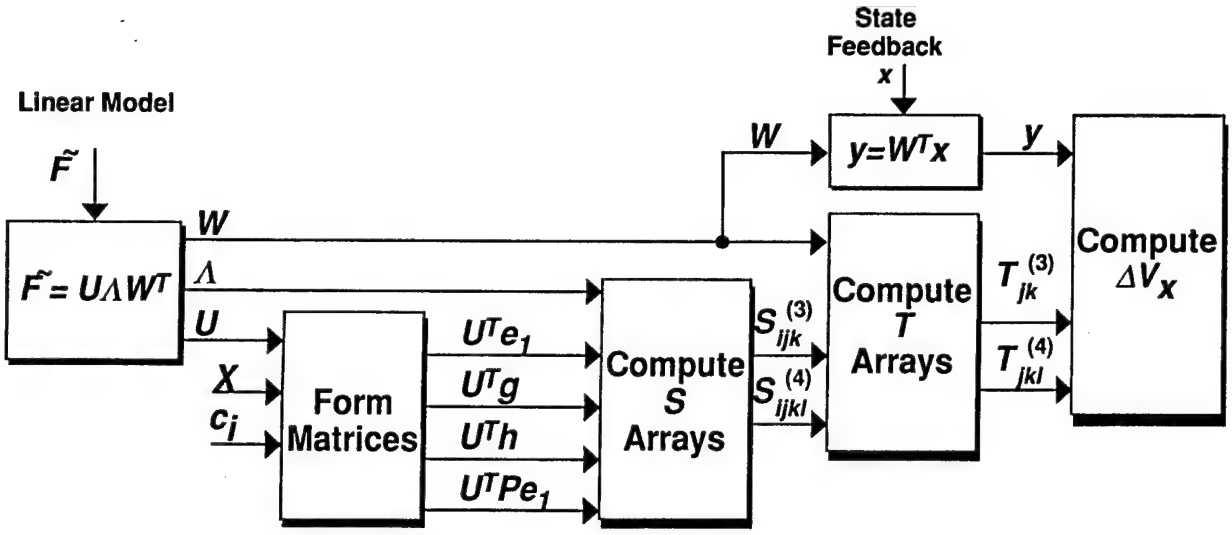


Figure 2.3 Effect of aerodynamic fin deflections on the pitching moment.



$$\Delta V_x^T(x) = \sum_j \sum_k T_{jk}^{(3)} y_j y_k + \sum_j \sum_k \sum_l T_{jkl}^{(4)} y_j y_k y_l$$

$$u(x) = [1 \;\; 0]\left(-R^{-1}\right)\left\{\tfrac{1}{2} B^T \left(2Xx + \Delta V_x^T\right) + S^T C x\right\}$$

Figure 2.4 Calculations used in computing $\Delta V_x^T(x)$.

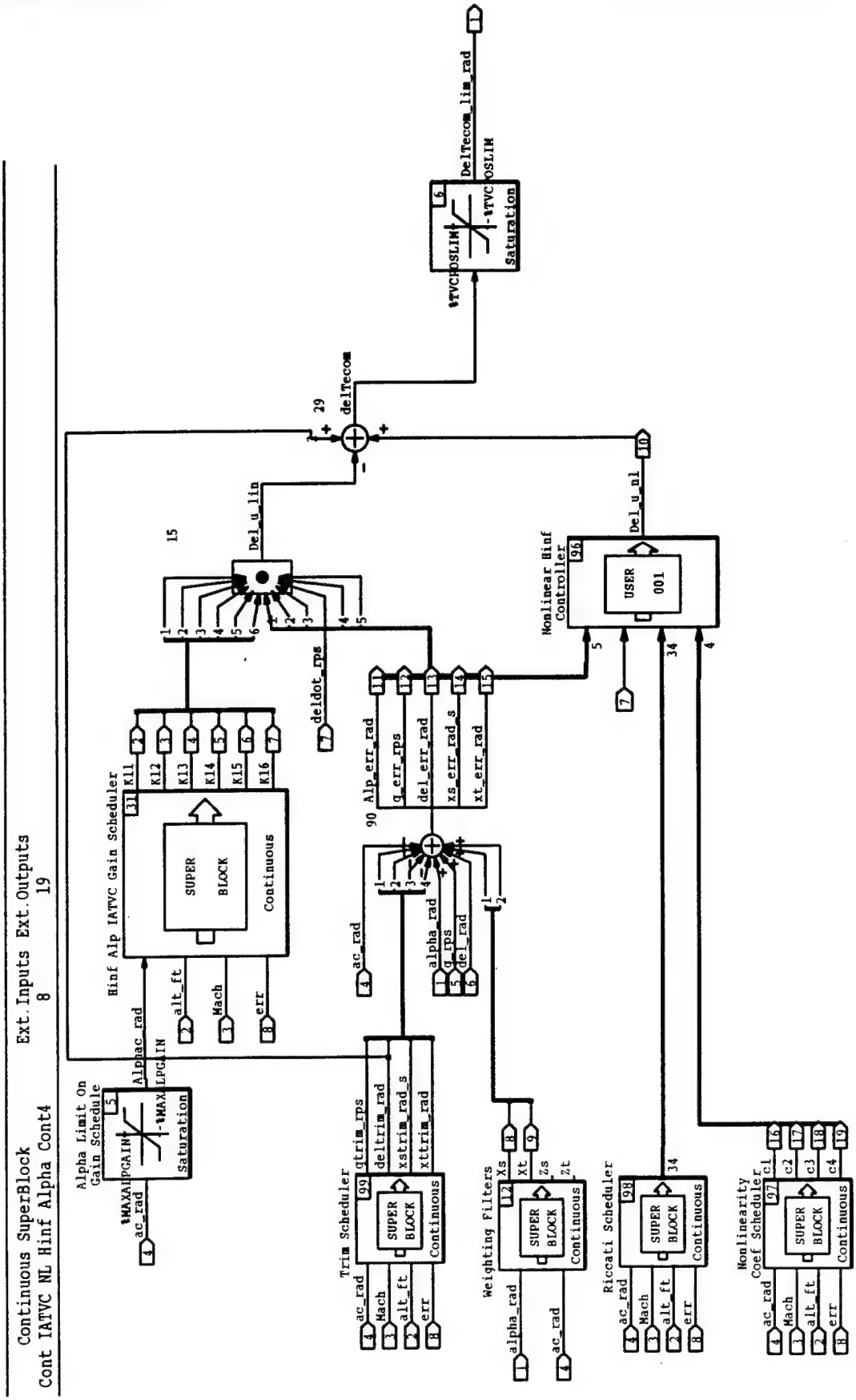


Figure 2.5 MATRIXx implementation of the nonlinear H_∞ control law.

contribution; and the nonlinear H_∞ contribution. The trim deflection is calculated in the Trim Scheduler superblock. The linear H_∞ contribution is calculated by looking up the linear gains in the Hinf Alp IATVC Gain Scheduler superblock, shown in the figure. The nonlinear H_∞ software described in Figure 2.4 is implemented in the User Code block User 001.

Figure 2.6 shows an animation of the simulation results. The figure shows an F-15 launch of the agile missile with missile trajectories for a 45 and 60 degree α command. The simulation stops when the missile's heading has changed by 180 degrees. The goal is to perform this heading change maneuver as quickly as possible.

Figure 2.7 shows time histories of important simulation variables. The α response follows the command as desired. The actuator time histories are well within the deflection and rate limits for the actuator.

Figure 2.7o shows the contribution to the control made by solving for the nonlinear part $\Delta V_x^T(x)$. This contribution is small when compared to the magnitude of the linear control. Since the linear gains are scheduled every 10 degrees AOA, the nonlinearities are captured (and adequately compensated for) in the linear gain table. This leads to a possible conclusion that if the linear design adequately covers the nonlinearities, then the contribution to the control by $\Delta V_x^T(x)$ will be small. This is the case here.

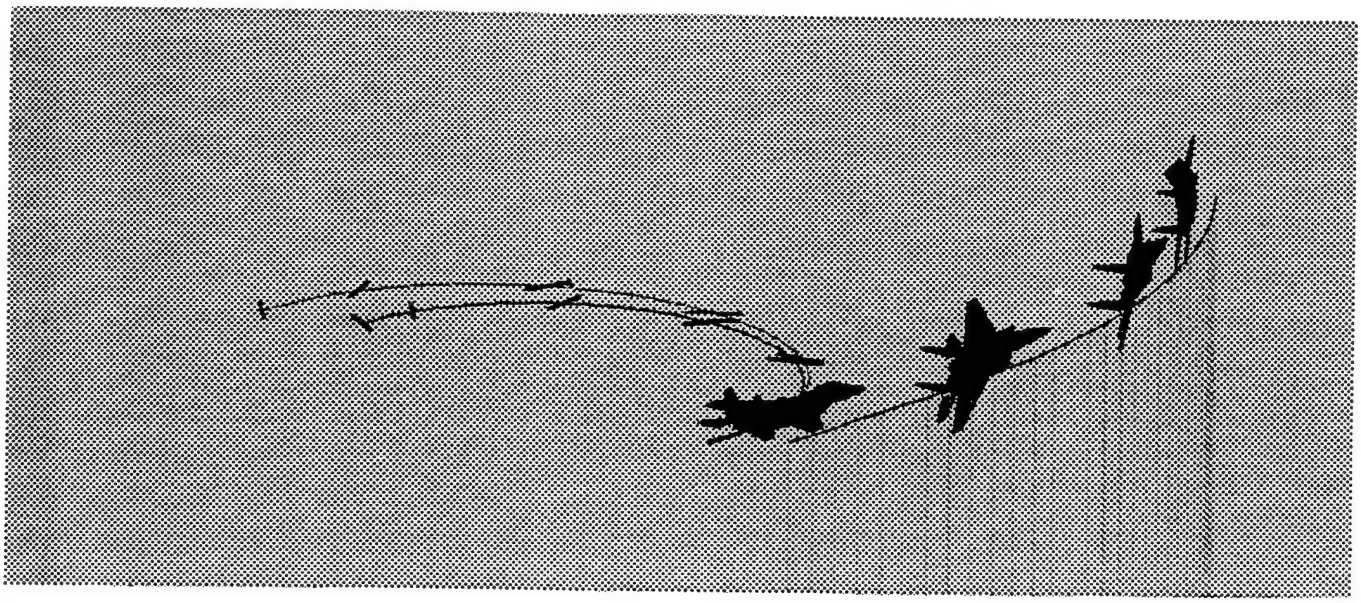


Figure 2.6 Animation of simulation results.

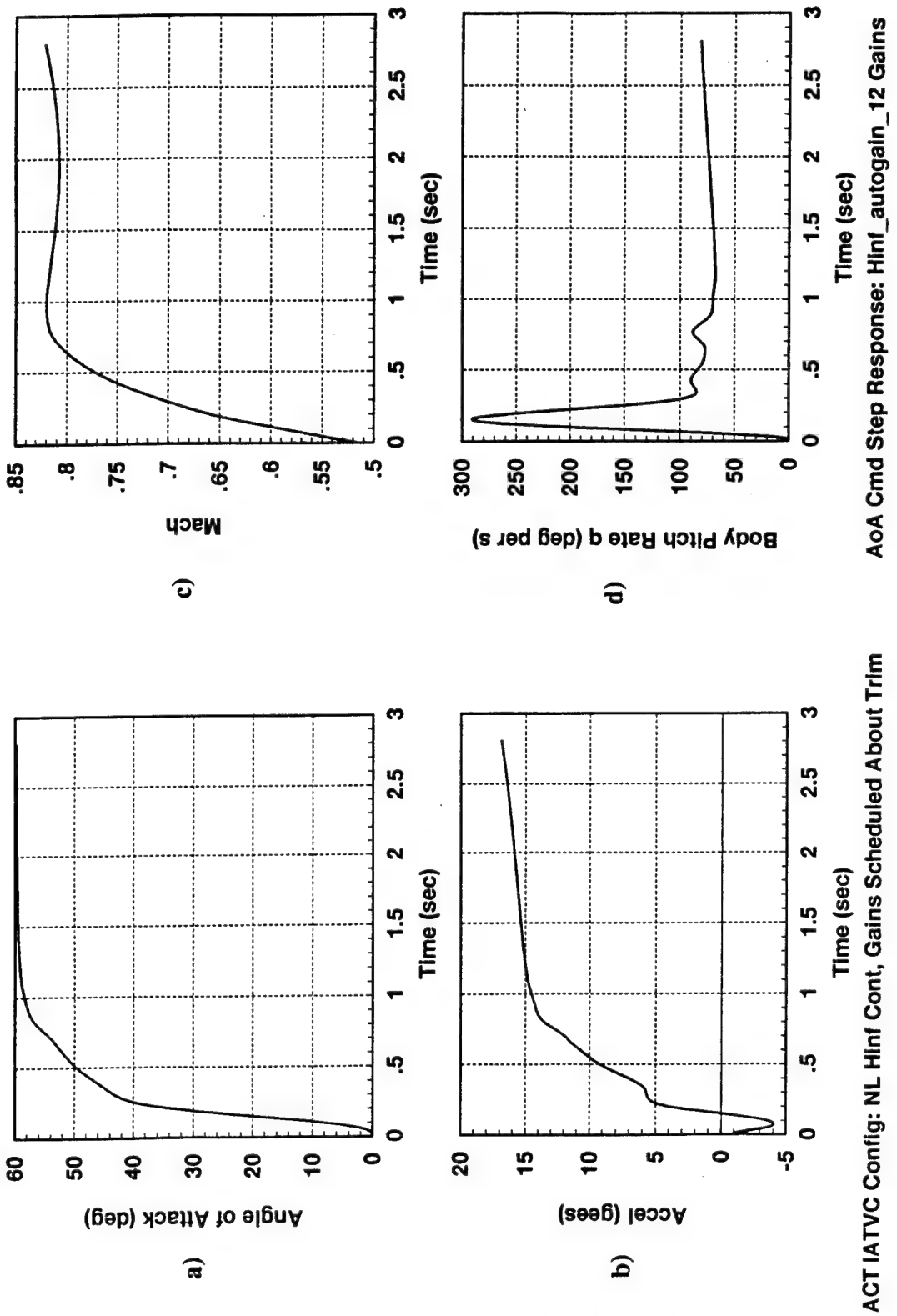


Figure 2.7 Time histories of important simulation variables.

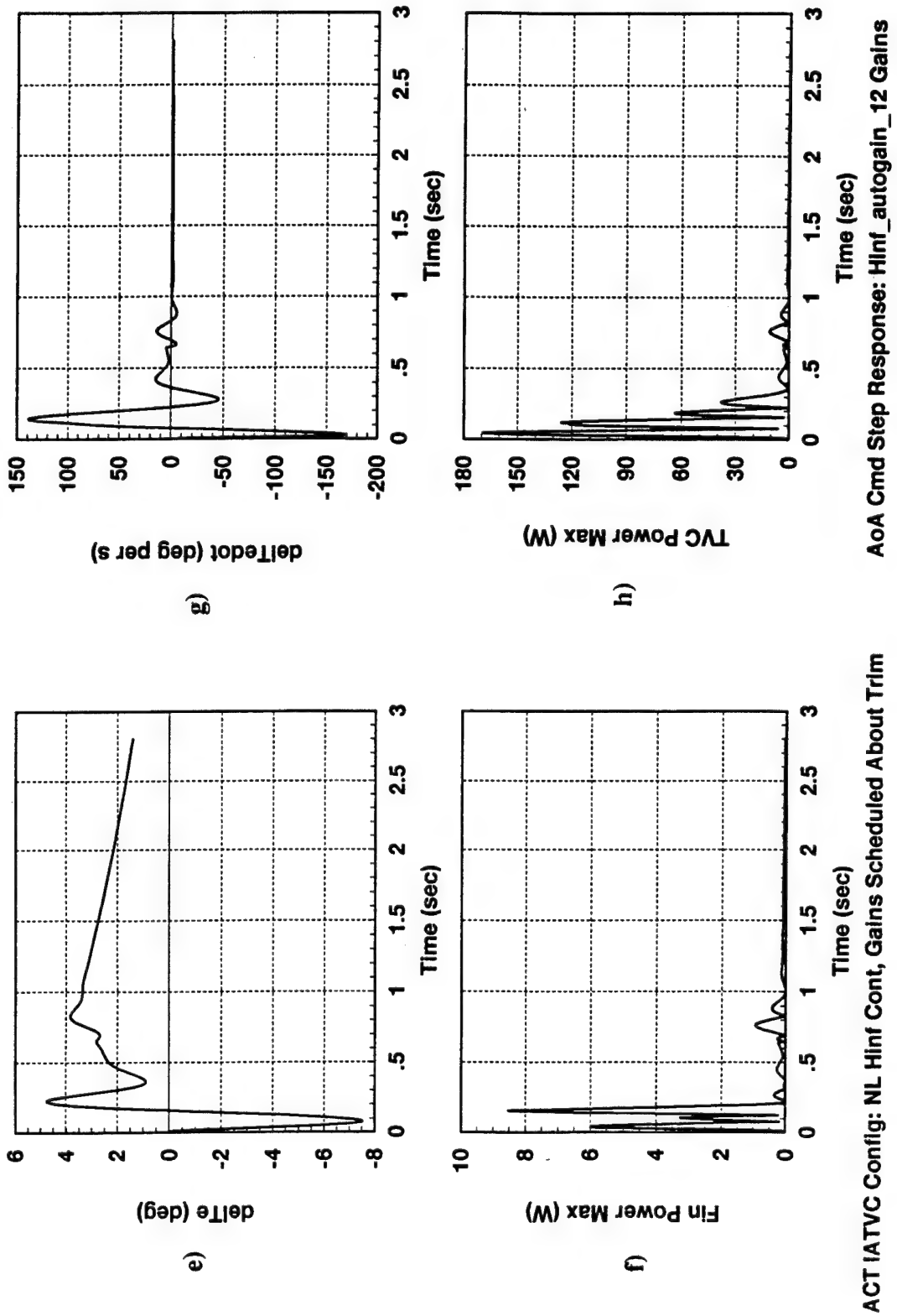


Figure 2.7 (Continued) Time histories of important simulation variables.

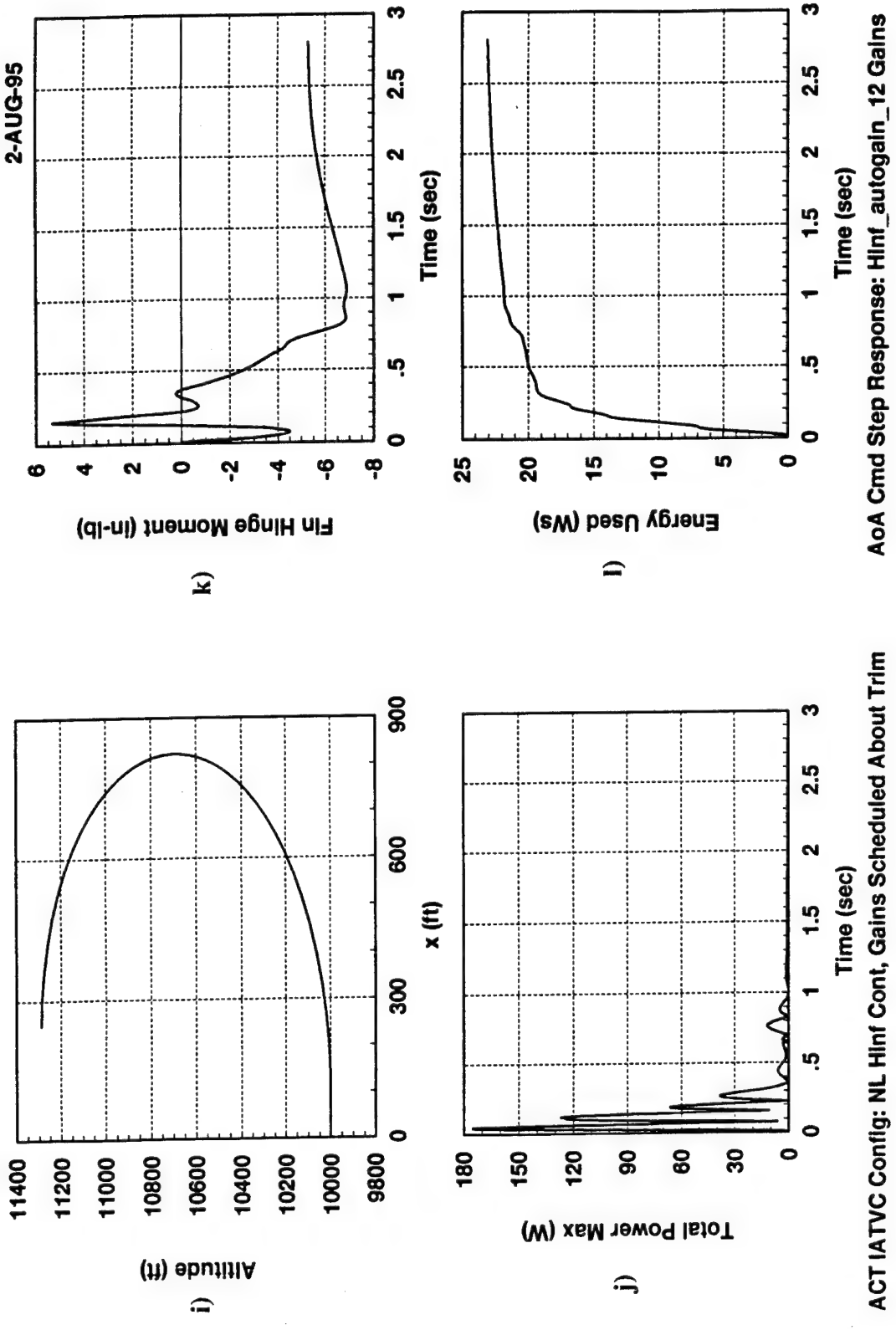


Figure 2.7 (Continued) Time histories of important simulation variables.

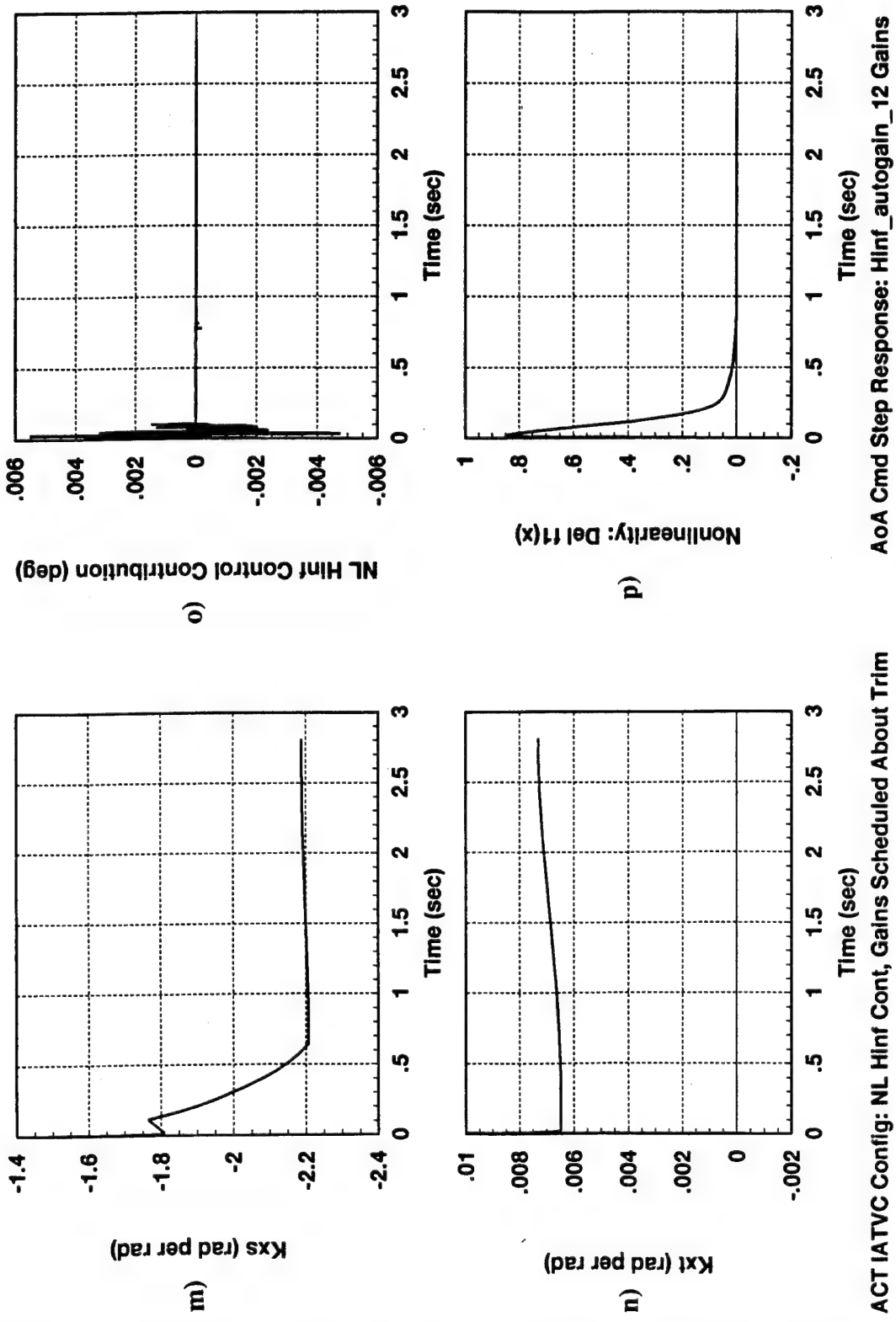


Figure 2.7 (Continued) Time histories of important simulation variables.

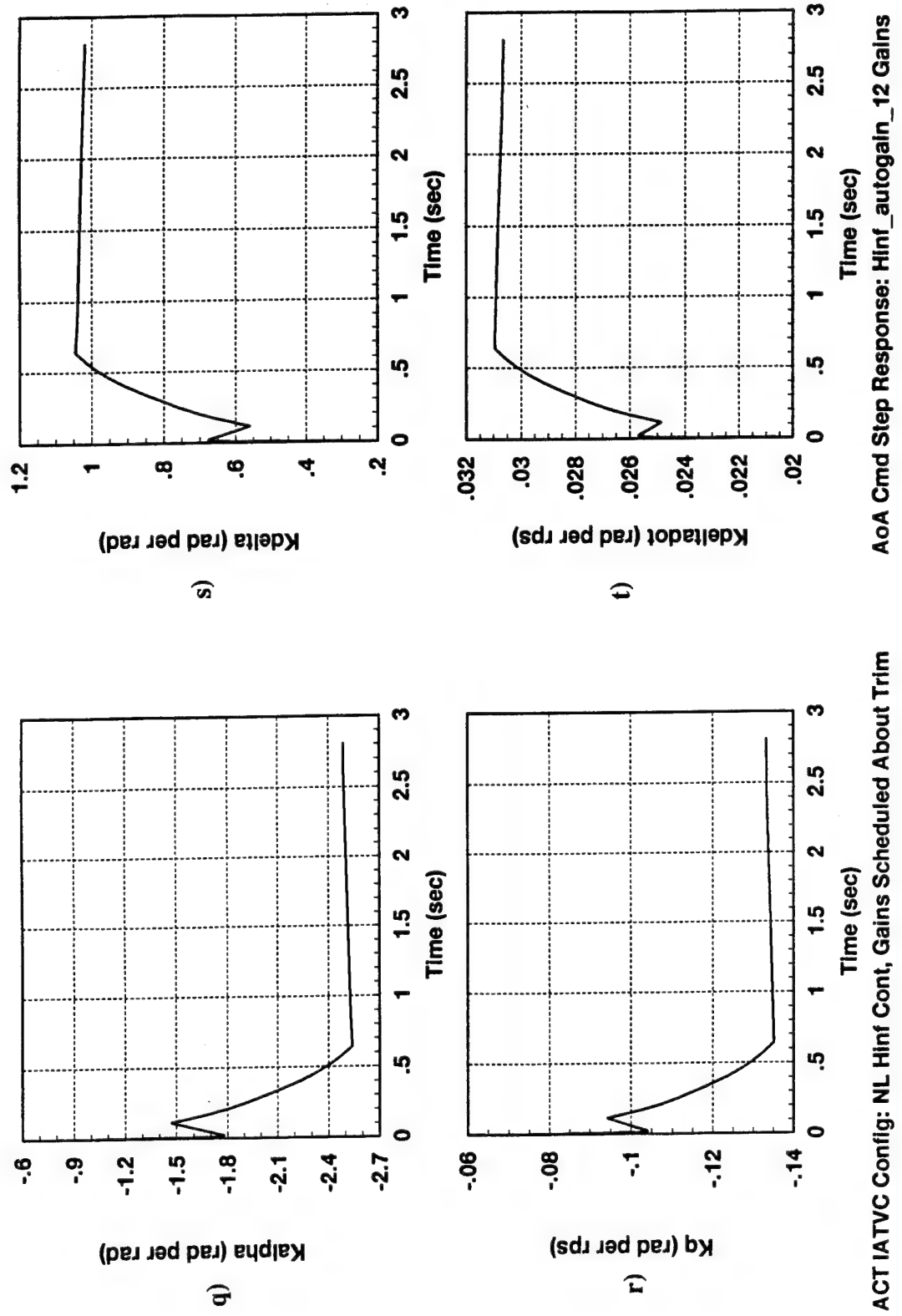


Figure 2.7 (Continued) Time histories of important simulation variables.

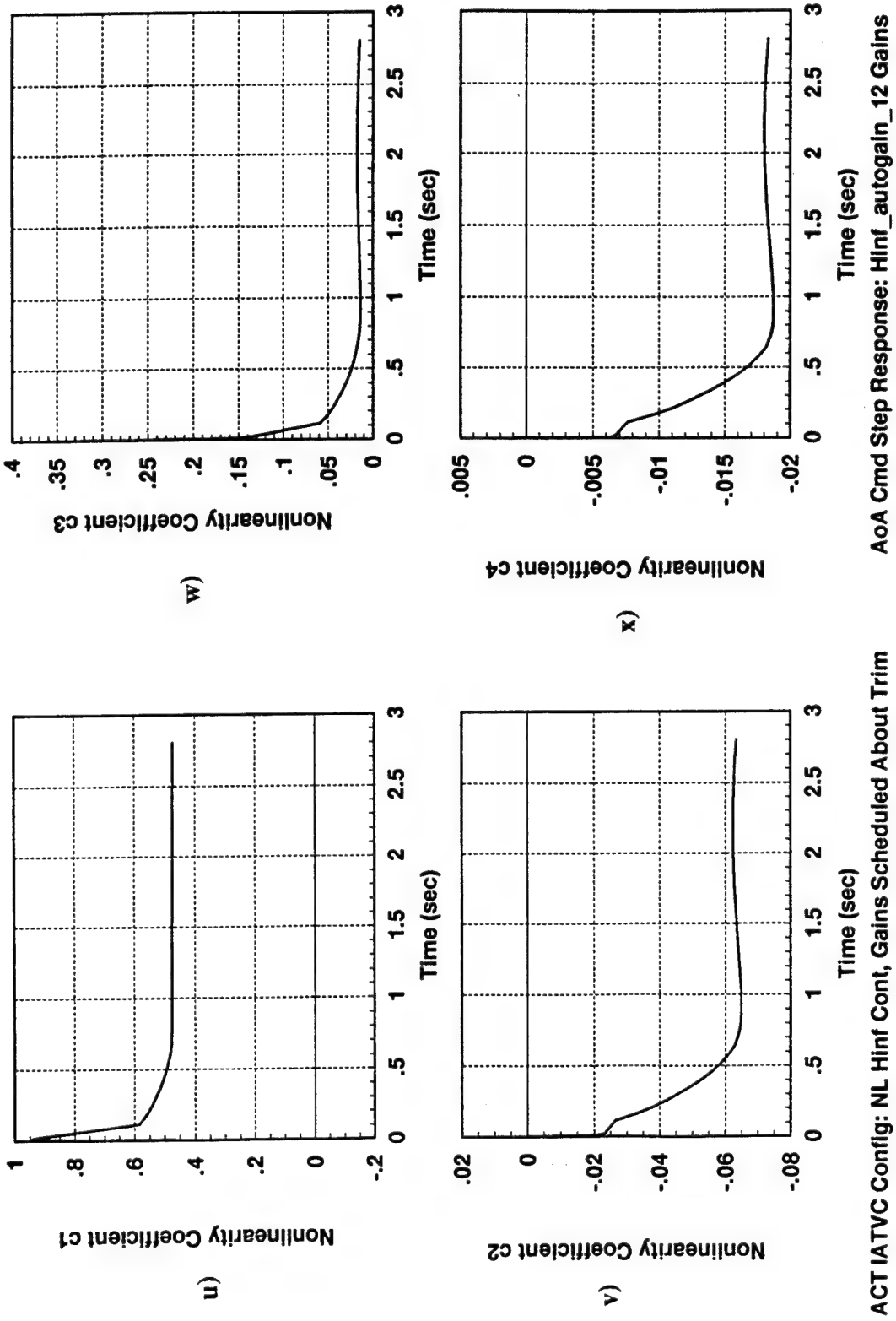


Figure 2.7 (Continued) Time histories of important simulation variables.

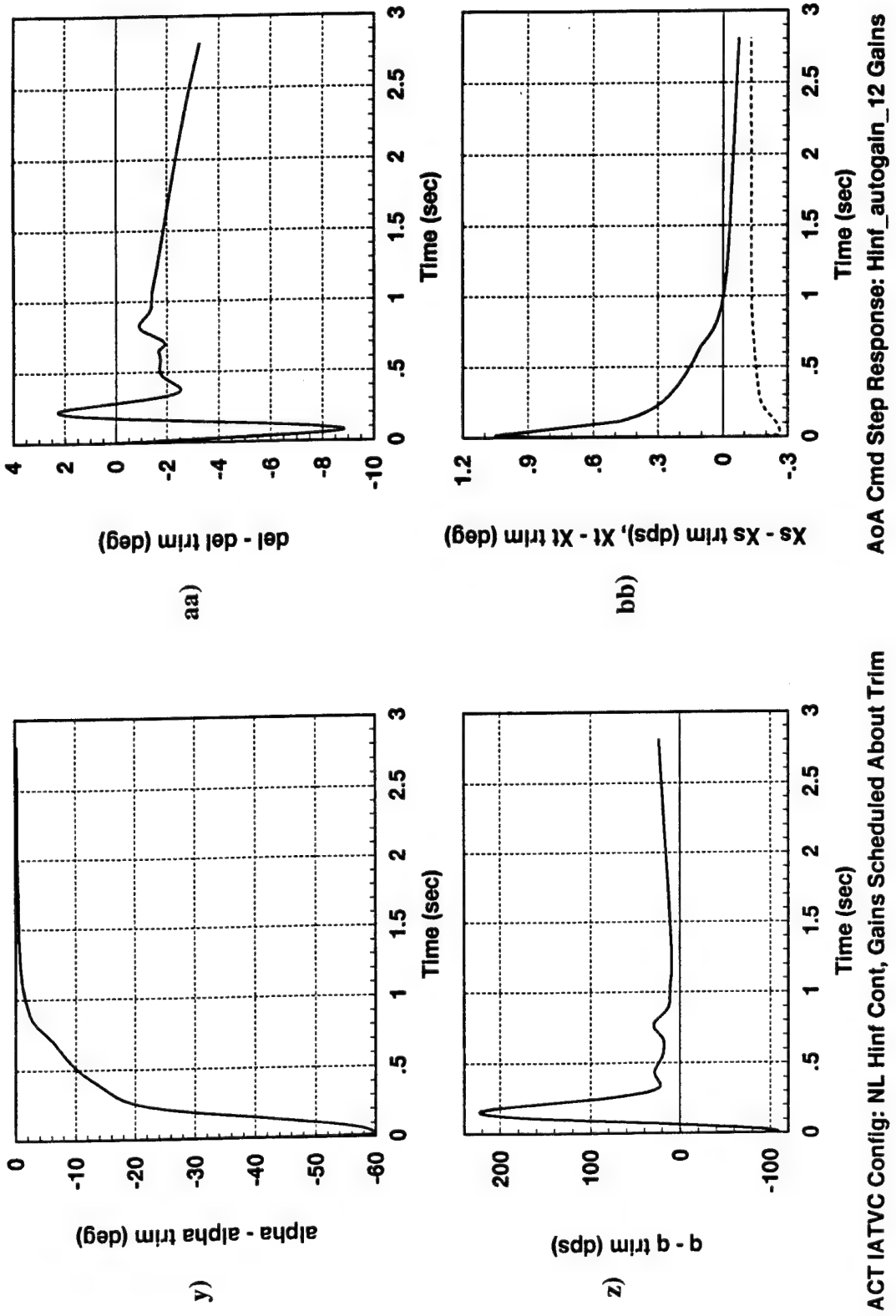


Figure 2.7 (Continued) Time histories of important simulation variables.

2.4 Conclusions and Future Research

A solution approach was presented for solving the Hamilton-Jacobi-Isaacs partial differential equation that arises in nonlinear H_∞ optimal control problems. The solution approach using successive approximation was applied to a missile flight control problem with six state variables. Although somewhat complicated, the state feedback control algorithms are implementable using standard software techniques. This solution approach has produced reasonable algorithms for solving the nonlinear partial differential equations that arise in the study of nonlinear H_∞ optimal control problems.

Nonlinear simulation was used to test the algorithms, and resulted in good performance. TVC nozzle deflections and rates commanded by the autopilot are well within limits. The angle of attack time history tracked the command very well.

Software for implementing the successive approximation solution procedure was developed that can calculate any number of successive approximations (only the first was used in this study). This software was documented (Appendix A) and is contained in Appendix B for application to other nonlinear control problems. The software can be obtained electronically by contacting the author at McDonnell Douglas (wisek@mdcgwy.mdc.com).

Future research will include a more detailed investigation into the use of gain scheduling to account for nonlinearities versus using our nonlinear H_∞ algorithms.

2.5 Chapter 2 References

[1]. van der Schaft, A. J., "On a state space approach to nonlinear H_∞ control," *Systems and Control Letters*, Vol. 16, 1991, pp. 1-8.

[2]. - ,"L_2-Gain analysis of nonlinear systems and nonlinear state feedback H_∞ control," *IEEE Trans. Automat. Contr.*, Vol. 37, 1992, pp. 770-784.

[3]. Isidori, A. and Astolfi, A., "Disturbance attenuation and H_∞-control via measurement feedback in nonlinear systems," *IEEE Trans. Automat. Contr*, Vol. 37, 1992pp. 1283-1293, pp. 1283-1293.

[4]. Doyle, J. C., Glover, K., Khargonekar, P. P., and Francis, B. A., "State space solutions to standard H_2 and H_∞ control problems," *IEEE Trans. Automat. Contr*, Vol. 34, 1989, pp. 831-846.

[5]. Basar, T., and Bernhard, P., *H_∞-Optimal Control and Related Minimax Design Problems, A Dynamic Game Approach.* Berlin: Birkhauser, 1990.

[6]. Ball, J. A., and Helton, J. W., "H_∞ control for nonlinear plants: Connection with differential games," in Proc. 28th Conf. Decision Contr., Tampa, FL, Dec. 1989, pp. 956-962.

[7]. Isaacs, R. , *Differential Games.* New York: John Wiley, 1965.

[8]. Ball, J. A., Helton, J. W., and Walker, M. L., "H_∞ Control for Nonlinear Systems with Output Feedback," *IEEE Trans. Automat. Contr.*, Vol. 38, 1993, pp. 546-559.

[9]. Ford, L. R., *Differential Equations*, McGraw Hill, New York, 1933.

[10]. Wassom, S.R., Faupell, L.C. , and Perley, T., "Integrated Aerofin/TVC for Tactical Missiles." *Journal of Propulsion and Power*, Vol. 7, 1991, pp. 374-381.

[11]. Wise, K. A., Mears, B.C., and Poolla, K., "Missile autopilot design using H_∞ optimal control with μ-synthesis," *Proc. of the American Control Conference,* San Diego, CA., May 1990, pp. 2362-2367.

3 Variable Structure Control Of Missiles

Recent studies have indicated that missile agility can provide a decided air superiority advantage when applied to close-in, low speed, high angle of attack, engagement scenarios. However, at low dynamic pressures, aerodynamic controls are not effective, and therefore missile agility requires some form of alternate control, such as reaction jets or thrust vectoring. The use of Reaction Control Valve (RCV) actuators and Thrust Vector Control (TVC) actuators can be challenging due to system nonlinearities resulting from hardware limitations. In particular, TVC and throttleable solid fuel propellant RCV systems have hard thrust magnitude and thrust rate limits as well as other actuator nonlinearities, while low cost RCV actuation may be provided by on-off valves which require discontinuous control.

This research addresses flight control system design for an agile missile with throttleable RCVs, with a focus on maximizing performance during an agile turn to the rear hemisphere. A pitch rate command autopilot topology was selected for this maneuver. In order to achieve the desired performance, the autopilot must be capable of very rapid command following. For a linear quadratic performance objective, this criteria requires a small weighting on the control variable, often called the "cheap control" optimization problem.

It is well known that the optimal feedback control for the "cheap control" Linear Quadratic Regulator (LQR) problem results in loops with high gain [1]. In this work we examine the "cheap control" H_∞ optimal control problem, and apply the resulting near optimum feedback control to the agile missile autopilot design problem. For comparison, the "cheap control" LQR problem is reviewed, and the near optimum regulator results are used to design a second missile autopilot for comparison.

In Young et al [2], it is shown that all high-gain systems can be represented as singularly perturbed systems, and therefore can be decomposed into slow and fast subsystems. Moreover, by the method of Chow and Kokotovic [3], the near optimum LQR design is a composition of the slow system regulator and the fast subsystem regulator. An advantage of the Chow and Kokotovic approach is that the controller can be designed independent of the singular perturbation parameter ε, therefore avoiding the problem of solving stiff differential equations. Recent developments [6] show that H_∞ optimal control problems and Linear Quadratic (LQ) differential games are closely related, and thus LQ game theory results can be used to develop worst-case H_∞ optimal controllers. Moreover, in [4], this relationship is extended to develop a method to design worst case H_∞ optimal controllers for singularly perturbed systems. In [4], Pan and Basar derive

conditions under which a composite controller exists and can be constructed independent of the singular perturbation parameter ε. However, the construction of the best approximate controller is more involved than the sum of the slow and fast controllers as in the LQR case.

3.1 High Gain Feedback Systems With Disturbance Terms

In this section we extend the results in [2] to show that under the standard assumptions, the high gain feedback system with disturbance term can be decomposed into slow and fast subsystems, where the fast transient occurs in the range of B_0 and the slow motion is confined to the nullspace of C_2.

Consider the linear time invariant high gain system of the form

$$\begin{aligned} \dot{x}_0 &= A_0 x_0 + B_0 u + D_0 w \\ w &= C_1 x_0 \\ u &= g C_2 x_0 \end{aligned} \tag{3.1}$$

where g is the high gain factor, the state $x_0 \in R^n$, the control $u \in R^m$, and the disturbance $w \in R^p$. Let ε be a small positive scalar such that, $\varepsilon = 1/g$, so that as g approaches infinity, ε approaches zero. By substituting for u and w in Eq. (3.1), and rewriting (3.1) in terms of ε, results in

$$\varepsilon \dot{x}_0 = (\varepsilon A_0 + B_0 C_2 + \varepsilon D_0 C_1) x_0 \tag{3.2}$$

We now show that if

$$rank(B_0 C_2) = rank(C_2 B_0) = m \tag{3.3}$$

and if the nonzero eigenvalues of $B_0 C_2$ have negative real parts

$$Re(\lambda_i(B_0 C_2)) < 0 \quad i = 1, \cdots, m \tag{3.4}$$

then the dynamical behavior of (3.1) is characterized by a fast transient to an $O(\varepsilon)$ neighborhood of $C_2 x_0 = 0$, followed by a slow motion in this neighborhood.

The system in Eq. (3.2) can be converted into the standard singularly perturbed form by letting

$$\tilde{x} = T x_0 \tag{3.5}$$

where the similarity transformation in Eq. (3.5) is two successive transformations, given by,

$$x = Mx_0 \qquad (3.6)$$
$$\tilde{x} = \Gamma x$$

The first transformation matrix M is applied to Eq. (3.1), and decouples the high gain control u from the first $(n-m)$ states resulting in the system

$$\dot{x}_1 = A_{11}x_1 + A_{12}x_2 + D_1 w \qquad (3.7)$$
$$\dot{x}_2 = A_{21}x_1 + A_{22}x_2 + B_2 u + D_2 w$$

where

$$MA_0M^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad MB_0 = \begin{bmatrix} 0 & B_2^T \end{bmatrix}^T$$

$$C_1M^{-1} = \begin{bmatrix} C_{11} & C_{12} \end{bmatrix} \quad C_2M^{-1} = \begin{bmatrix} C_{21} & C_{22} \end{bmatrix} \quad MD_0 = \begin{bmatrix} D_1^T & D_2^T \end{bmatrix}^T$$

Note that the control u only enters into the x_2 dynamics. In Weil and Wise [7] we extended this approach to include a slow control (not high gain) that enters only into the x_1 dynamics. This was used to blend aero controls (slow) with RCV controls (fast).

Next, substitute the feedback control $u = gC_2M^{-1}x$ into Eq. (3.7). The transformation matrix Γ in Eq. (3.6) represents the coordinate change given by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \xrightarrow{\Gamma} \begin{bmatrix} z \\ y \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} I_{n-m} & 0 \\ C_{21} & C_{22} \end{bmatrix} \qquad (3.8)$$

and transforms Eq. (3.7) into the system described by

$$\dot{z} = F_{11}z + F_{12}y + G_1 w \qquad (3.9)$$
$$\varepsilon\dot{y} = \varepsilon H_1 z + (C_2B_0 + \varepsilon H_2)y + \varepsilon G_2 w$$

where

$$
\begin{aligned}
F_{11} &= A_{11} - A_{12}C_{22}^{-1}C_{21} \\
F_{12} &= A_{12}C_{22}^{-1} \\
H_1 &= (C_{21}A_{11} + C_{22}A_{21})C_{22}^{-1}C_{21} + H_2 \\
H_2 &= (C_{21}A_{12} + C_{22}A_{22})C_{22}^{-1} \\
G_1 &= D_1 \\
G_2 &= C_{21}D_1 + C_{22}D_2
\end{aligned}
$$

This transformation (Eq. (3.8)) separates the slow (z) and fast (y) states, however, they still remain coupled. To eliminate the slow state from the $\dot{y}$ equation, the following transformation

$$\eta = y + \varepsilon L z \tag{3.10}$$

from Pan and Basar [4] and Kokotovic and Haddad [5] is applied to Eq. (3.9) to obtain

$$
\begin{aligned}
\dot{z} &= \left(F_{11} - \varepsilon F_{12}(L_0 + \varepsilon E_\varepsilon)\right)z + F_{12}\eta + G_1 w \\
\varepsilon\dot{\eta} &= \left(C_2B_0 + \varepsilon H_2 + \varepsilon^2(L_0 + \varepsilon E_\varepsilon)F_{12}\right)\eta + \left(\varepsilon G_2 + \varepsilon^2(L_0 + \varepsilon E_\varepsilon)G_1\right)w
\end{aligned} \tag{3.11}
$$

where

$$
\begin{aligned}
L &= (C_2B_0 + \varepsilon H_2)^{-1}H_1 \\
&= L_0 + \varepsilon E_\varepsilon \\
E_\varepsilon &= (C_2B_0)^{-2}H_1F_{11} + O(\varepsilon)
\end{aligned} \tag{3.12}
$$

The slow and fast eigenvalues of the system in Eq. (3.11) are given by

$$
\begin{aligned}
\lambda_j^s &= \lambda_j(F_{11}) + O(\varepsilon) \quad j = 1,\cdots,n-m \\
\lambda_i^f &= \varepsilon^{-1}\left(\lambda_i(C_2B_0) + O(\varepsilon)\right) \quad i = 1,\cdots,m
\end{aligned} \tag{3.13}
$$

By choosing ε sufficiently small, (with the assumptions in Eqs. (3.3) and (3.4) satisfied), the fast subsystem is asymptotically stable.

Equation (3.13) clarifies the two time scale property of the high gain system (3.2). The fast dynamics decay exponentially on the time scale t/ε, while the slow dynamics evolve on the time scale t.

Asymptotic stability of the fast dynamics is guaranteed by Eq. (3.4). Asymptotic stability of the slow dynamics (assuming that the disturbance is independent of the state) requires that the eigenvalues of F_{11} reside in the open left half of the complex plane.

With the disturbance w given by Eq. (3.1), Eq. (3.9) becomes

$$\begin{aligned} \dot{z} &= \tilde{F}_{11} z + \tilde{F}_{12} y \\ \varepsilon \dot{y} &= \varepsilon \tilde{H}_1 z + \left(C_2 B_0 + \varepsilon \tilde{H}_2 \right) y \end{aligned} \tag{3.14}$$

where

$$\begin{aligned} \tilde{F}_{11} &= F_{11} + G_1 \left(C_{11} - C_{12} C_{22}^{-1} C_{21} \right) \\ \tilde{F}_{12} &= F_{12} + G_1 C_{12} C_{22}^{-1} \\ \tilde{H}_1 &= H_1 + G_2 \left(C_{11} - C_{12} C_{22}^{-1} C_{21} \right) \\ \tilde{H}_2 &= H_2 + G_2 C_{12} C_{22}^{-1} \end{aligned}$$

By applying the transformation in Eq. (3.10), the equivalent expression for Eq. (3.11) is

$$\begin{aligned} \dot{z} &= \left(\tilde{F}_{11} - \varepsilon \tilde{F}_{12} \left(\tilde{L}_0 + \varepsilon \tilde{E}_\varepsilon \right) \right) z + \tilde{F}_{12} \eta \\ \varepsilon \dot{\eta} &= \left(C_2 B_0 + \varepsilon \tilde{H}_2 + \varepsilon^2 \left(\tilde{L}_0 + \varepsilon \tilde{E}_\varepsilon \right) \right) \eta \end{aligned} \tag{3.15}$$

where

$$\begin{aligned} \tilde{L} &= \left(C_2 B_0 + \varepsilon \tilde{H}_2 \right)^{-1} \tilde{H}_1 \\ &= \tilde{L}_0 + \varepsilon \tilde{E}_\varepsilon \\ \tilde{E}_\varepsilon &= \left(C_2 B_0 \right)^{-2} \tilde{H}_1 \tilde{F}_{11} + O(\varepsilon) \end{aligned}$$

The eigenvalues of Eq. (3.15) are given by

$$\begin{aligned} \lambda_j^s &= \lambda_j \left(\tilde{F}_{11} \right) + O(\varepsilon) \quad j = 1, \cdots, n-m \\ \lambda_i^f &= \varepsilon^{-1} \left(\lambda_i \left(C_2 B_0 \right) + O(\varepsilon) \right) \quad i = 1, \cdots, m \end{aligned} \tag{3.16}$$

which shows that the disturbance does not affect the fast subsystem eigenvalues, however, the stability of the slow subsystem is dependent upon the disturbance feedback gains.

It has been shown that the eigenvalues of the high gain system (with $w=0$) are the transmission zeros of the open loop system with output $y=C_2x_0$. Therefore, the eigenvalues of F_{11} are the transmission zeros. This result is used in [2] to develop a process to calculate transmission zeros. In the case where $w=C_1x_0$, (Eq. (3.1)), the eigenvalues of $\tilde{F}_{11}$ are the open loop transmission zeros of with the output $y=C_2x_0$.

The results of [2] show that T decomposes the system in Eq. (3.1) (with $w=0$) into the null space of C_2 and the range space of B_0. Our results, with $w=C_1x_0$, show that the system

$$\dot{\tilde{x}}=T\dot{x}_0 \tag{3.17}$$

is given by Eq. (3.15). Partitioning the transformation matrices as

$$M=\begin{bmatrix}M_1\\M_2\end{bmatrix} \qquad M^{-1}=\begin{bmatrix}S_1 & S_2\end{bmatrix} \tag{3.18}$$

where M_1, S_1^T are $(n-m)\times n$, and M_2, S_2^T are $m\times n$ matrices, the vector x_0 can be written as

$$x_0=\left(S_1-S_2C_{22}^{-1}C_{21}\right)z+B_0\left(C_{22}B_0\right)^{-1}y \tag{3.19}$$

where $0=C_2\left(S_1-S_2C_{22}^{-1}C_{21}\right)$.

Let

$$N=S_1-S_2C_{22}^{-1}C_{21} \tag{3.20}$$

then $C_2N=0$, $y=C_2x_0$, and $z=M_1x_0$, where $M_1B_0=0$.

3.2 Linear Quadratic High Gain Feedback Control

Consider the system and performance index described by

$$\begin{aligned}\dot{x}_0&=A_0x_0+B_0u\\ J&=\frac{1}{2}\int_0^\infty\left(x_0^TQ_0x_0+\varepsilon^2u^TRu\right)d\tau\end{aligned} \tag{3.21}$$

It is well known that high-gain feedback control can result from the linear quadratic (LQ) performance index having a small penalty, $\varepsilon>0$, on u. This so-called "cheap control" optimization problem lightly penalizes the control, resulting in rapid regulation of the states

weighted by the penalty matrix Q_0. O'Malley and Jameson [1] (among others) have studied the "cheap control" problem and have produced detailed results for the case when $B_0^T Q_0 B_0 > 0$. As noted in [2], this assumption implies the condition in Eq. (3.3), and therefore the optimal "cheap control" state regulator possesses the two time scale property demonstrated in the previous section. Moreover, a near optimum high-gain state regulator can be designed (independent of ε) using the method of [3]. Instead of solving the full order LQ optimization, the following two reduced order regulator problems are solved.

The Slow Regulator Problem:

Consider the slow system dynamics given by

$$\dot{x}_s = A_{11} x_s + A_{12} u_s \tag{3.22}$$

with performance index

$$J = \frac{1}{2} \int_0^\infty \left(x_s^T Q_{11} x_s + 2 x_s^T Q_{12} u_s + u_s^T Q_{22} u_s \right) d\tau \tag{3.23}$$

The Fast Regulator Problem:

Consider the fast system dynamics given by

$$\dot{x}_f = B_2 u_f \tag{3.24}$$

with performance index

$$J = \frac{1}{2} \int_0^\infty \left(x_f^T Q_{22} x_f + u_f^T R u_f \right) d\tau \tag{3.25}$$

where A_{11}, A_{12}, B_2, are obtaned using the transformation M as in Eq. (3.7), with the slow state vector x_s an $(n-m)$ vector, x_f, u_f, u_s, are m vectors, and Q_{ij} are the submatrices of $Q = \left(M^{-1}\right)^T Q_0 M^{-1}$ where M is as in Eqs. (3.6) and (3.7).

Lemma 3.1:

If the pair (A_0, B_0) (Eq. ((3.21)) is stabilizable, then the pair (A_{11}, A_{12}) is stabilizable

Proof: (see [2])

Lemma 3.2:

If the pair (A_0, B_0) is stabilizable, the pair (A_{11}, Y) is detectable, where

$$Y^T Y = Q_{11} - Q_{12} Q_{22}^{-1} Q_{12}^T \tag{3.26}$$

and $B_0^T Q_0 B_0 > 0$, Then, there exists a unique stabilizing solution P_s of the Algebraic Riccati Equation (ARE)

$$P_s\left(A_{11} - A_{12} Q_{22}^{-1} Q_{12}^T\right) + \left(A_{11} - A_{12} Q_{22}^{-1} Q_{12}^T\right)^T P_s + Y^T Y - P_s A_{12} Q_{22}^{-1} A_{12}^T P_s = 0 \tag{3.27}$$

and the optimal control for the slow subsystem is given by

$$\begin{aligned} u_s &= -Q_{22}^{-1}\left(Q_{12}^T + A_{12}^T P_s\right) x_s \\ &= -K_s x_s \end{aligned} \tag{3.28}$$

Proof: (see [2])

The optimal feedback control for the fast subsystem is given by

$$\begin{aligned} u_f &= -R^{-1} B_2 P_f x_f \\ &= -K_f x_f \end{aligned} \tag{3.29}$$

where P_f is positive definite and given by

$$P_f = W^{-1}(W Q_{22} W)^{1/2} W^{-1} \tag{3.30}$$

with $W = \left(B_2 R^{-1} B_2^T\right)^{1/2}$.

The composite control (combining slow and fast controls) is given by

$$u_c = -g\left(K_f K_s x_1 + K_f x_2\right) \tag{3.31}$$

Theorem 3.1:

Under the conditions of Lemma 3.2, the composite control u_c is near optimal in the sense that the performance J of the system described by Eq. (3.7) using (3.31) is $O\left(\varepsilon^2\right)$ close to its optimal performance.

Proof: (see [2]).

Notice that the stabilizability and detectability conditions required for the solution of the standard LQR problem are replaced by the conditions of Lemma 3.2. Moreover, the stabilizability and detectability conditions are equivalent to the existence of an unique positive semidefinite solution P_s to Eq. (3.27) that renders $A_{11} - A_{12}K_s$ Hurwitz. To regulate the output $y = C_2 x_0$, Q is chosen as $C_2^T C_2$. If $rank(C_2) = m$, then the near optimal control calculated by letting $y_s = C_{21} x_s$, $y_f = C_{22} x_f$ in Eqs. (3.22) through (3.25), respectively, is $O(\varepsilon^2)$ close to the optimal control. The detectability condition of Lemma 3.2 is replaced by the stabilizability of (A_{11}, A_{12}) which implies that $Re\left(\lambda\left(A_{11} - A_{12}C_{22}^{-1}C_{21}\right)\right) < 0$.

As noted in [2], the solution to (3.27) is $P_s = 0$. Therefore,

$$u_s = -Q_{22}^{-1}Q_{12}^T x_s \tag{3.32}$$

Therefore, the transmission zeros of the system in Eq. (3.7) with output $y = C_2 x_0$ must be in the open left half plane.

3.3 High Gain H_∞ Optimal Control

In this section the results of Pan and Basar [4] are used to show that high-gain feedback control can result from the optimization of the system in Eq. (3.1) with respect to a quadratic cost function having a small penalty ε on u, and disturbance bound $\gamma >> \varepsilon$. Consider the performance index

$$J_\gamma = \tfrac{1}{2}\int_0^\infty \left(x_0^T Q_0 x_0 + \varepsilon^2 u^T R u - \gamma^2 w^T w\right) d\tau \tag{3.33}$$

For $\varepsilon > 0$ and small, $\gamma >> \varepsilon$, and $B_0^T Q_0 B_0 > 0$, the system is high gain and possesses the two time scale property previously discussed. As a result, a near optimum regulator can be designed as the composition of the slow subsystem H_∞ state regulator, and the fast subsystem LQ regulator (since no disturbance term appears in the fast dynamics).

Let $\gamma^*(\varepsilon)$ denote the smallest value of $\gamma > 0$ under which the differential game (Eq. (3.1) and Eq. (3.33)) has a bounded upper value when the control u is a closed loop state feedback control policy. Then, for $\varepsilon > 0$ and $\gamma > \gamma^*(\varepsilon)$, it is known [4] that this differential game has a saddle point solution given by

$$u^* = -\varepsilon^{-2} B_0^T R^{-1} Z x_0$$
$$w^* = \gamma^{-2} D_0^T Z x_0 \tag{3.34}$$

where Z is the solution to the ARE described by

$$A_0^T Z + Z A_0 - Z\left(\varepsilon^{-2} B_0 R^{-1} B_0^T - \gamma^{-2} D_0 D_0^T\right) Z + Q_0 = 0 \tag{3.35}$$

Assume that the system (Eq. (3.1) has been transformed into the form given in Eq. (3.7). Using this same partition, define A, Q, B, and D as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ B_2 \end{bmatrix} \quad D = \begin{bmatrix} D_1 \\ 0 \end{bmatrix} \tag{3.36}$$

then Eq. (3.35) becomes

$$A^T Z + Z A - Z\left(\varepsilon^{-2} B R^{-1} B^T - \gamma^{-2} D D^T\right) Z + Q = 0 \tag{3.37}$$

Let Z be partitioned as

$$Z = \begin{bmatrix} Z_{11} & \varepsilon Z_{12} \\ \varepsilon Z_{12}^T & \varepsilon Z_{22} \end{bmatrix} \tag{3.38}$$

By substituting this into Eq. (3.37) and letting $\varepsilon \to 0$, the following algebraic equations are obtained:

$$A_{11}^T Z_{11} + Z_{11} A_{11} + \gamma^{-2} Z_{11} D_1 D_1^T Z_{11} + Q_{11} = Z_{12} B_2 R^{-1} B_2^T Z_{12}^T$$
$$Z_{11} A_{12} + Q_{12} = Z_{12} B_2 R^{-1} B_2^T Z_{22} \tag{3.39}$$
$$Q_{22} = Z_{22} B_2 R^{-1} B_2^T Z_{22}$$

Solving for Z_{22} and Z_{12} results in

$$Z_{22} = W^{-1} (W Q_{22} W)^{1/2} W^{-1}$$
$$Z_{12} = (Z_{11} A_{12} + Q_{12}) Z_{22}^{-1} W^{-2} \tag{3.40}$$

where $W = \left(B_2 R^{-1} B_2^T\right)^{1/2}$. Substituting these expressions into the Z_{11} expression results in

$$\left(A_{11}-A_{12}Q_{22}^{-1}Q_{12}^{T}\right)^{T}Z_{11}+Z_{11}\left(A_{11}-A_{12}Q_{22}^{-1}Q_{12}^{T}\right)- \\ Z_{11}\left(A_{12}Q_{22}^{-1}A_{12}^{T}-\gamma^{-2}D_1D_1^{T}\right)Z_{11}+Q_{11}-Q_{12}Q_{22}^{-1}Q_{12}^{T}=0 \tag{3.41}$$

Now, define the set

$$\Gamma=\{\bar{\gamma}>0:\forall\gamma>\bar{\gamma}\ \text{Eq. (3.41) has a bounded solution}\ Z_{11}>0,\ \text{and} \\ S=A_{11}-A_{12}Q_{22}^{-1}Q_{12}^{T}-\left(A_{12}Q_{22}^{-1}A_{12}^{T}-\gamma^{-2}D_1D_1^{T}\right)Z_{11}\ \text{is stable}\ \} \tag{3.42}$$

Let $\gamma^{*}=inf(\gamma\in\Gamma)$, then a solution to Eq. (3.41) exists for all $\gamma>\gamma^{*}$. Notice that Γ is nonempty since γ can be chosen sufficiently large. That is, in the limit as $\gamma\to\infty$, Eq. (3.41) $\to$ Eq. (3.27). Therefore, given that Lemma 3.2 holds for all $\gamma>\gamma^{*}$, a near optimum high gain state feedback regulator can be designed by solving the following two reduced order regulator (one H_∞, one LQ) problems.

The Slow H_∞ State Feedback Problem:

Using Eq. (3.7), with u defined in Eq. (3.1), and $\varepsilon=0$, we obtain the slow system dynamics

$$\dot{x}_s=A_{11}x_s+A_{12}u_s+D_1w_s \\ u_s=-C_{22}^{-1}C_{21}x_s \tag{3.43}$$

where

$$x_s=x_1 \\ \bar{x}_2=x_2(\varepsilon=0) \\ u_s=\bar{x}_2$$

The slow cost function is

$$J_{\gamma_s}=\frac{1}{2}\int_0^{\infty}\left(x_s^{T}Q_{11}x_s+x_s^{T}Q_{12}u_s+u_s^{T}Q_{12}^{T}x_s+u_s^{T}Q_{22}u_s-\gamma^{-2}w_s^{T}w_s\right)d\tau \tag{3.44}$$

Notice that u_s depends only on the slow state. Therefore, in contrast to [4], there is only one slow game to be considered. To convert Eq. (3.44) to the standard form with no crossterms, define

$$\bar{u}_s=u_s+Q_{22}^{-1}Q_{12}^{T}x_s \tag{3.45}$$

Substituting Eq. (3.45) into Eqs. (3.43) and (3.44), yields

$$\begin{aligned}\dot{x}_s &= \left(A_{11} - A_{12}Q_{22}^{-1}Q_{12}^T\right)x_s + A_{12}\bar{u}_s + D_1 w_s \\ J_{\gamma_s} &= \tfrac{1}{2}\int_0^\infty \left(x_s^T Q_{11} x_s + \bar{u}_s^T Q_{22}\bar{u}_s - \gamma^{-2} w_s^T w_s\right)d\tau\end{aligned} \tag{3.46}$$

Solution of this standard LQ differential game depends on the solution of the following ARE

$$\left(A_{11} - A_{12}Q_{22}^{-1}Q_{12}^T\right)^T Z_s + Z_s\left(A_{11} - A_{12}Q_{22}^{-1}Q_{12}^T\right) - Z_s\left(A_{12}Q_{22}^{-1}A_{12}^T - \gamma_s^{-2}D_1 D_1^T\right)Z_s + Q_{11} = 0 \tag{3.47}$$

Define

$$\begin{aligned}\Gamma_s = \{\gamma' > 0 : \forall \gamma > \gamma' \text{ Eq. (3.47) has a bounded solution } Z_s > 0, \text{ and} \\ Re\left(\lambda\left(A_{11} - A_{12}Q_{22}^{-1}Q_{12}^T - \left(A_{12}Q_{22}^{-1}A_{12}^T - \gamma^{-2}D_1 D_1^T\right)Z_s\right)\right) < 0\}\end{aligned} \tag{3.48}$$

Let

$$\gamma_s = inf(\gamma \in \Gamma_s) \tag{3.49}$$

Then, the transformed game has a bounded upper value if $\gamma > \gamma_s$, and only if $\gamma \geq \gamma_s$ [6]. For $\gamma > \gamma_s$, let Z_s be the unique nonnegative definite solution of Eq. (3.47). Then, there exist feedback saddle-point policies for the transformed game, given by

$$\begin{aligned}\bar{u}_{s_\gamma} &= -Q_{22}^{-1}A_{12}^T Z_s x_s \\ w_{s_\gamma} &= \gamma^{-2}D_1^T Z_s x_s\end{aligned} \tag{3.50}$$

Transforming Eq. (3.50) back into the original coordinates yields

$$\begin{aligned}u_{s_\gamma} &= -Q_{22}^{-1}\left(Q_{12}^T + A_{12}^T Z_s\right)x_s \\ &= -K_{1s}x_s \\ w_{s_\gamma} &= -K_{2s}x_s\end{aligned} \tag{3.51}$$

The Fast H_∞ State Feedback Problem

Define the fast state, x_f, the fast control, u_f, and the fast disturbance, w_f, as

$$
\begin{aligned}
x_f &= x_2 - \bar{x}_2 \\
u_f &= \varepsilon u \\
w_f &= 0
\end{aligned}
\tag{3.52}
$$

Then the fast subsystem dynamics and the associated cost function are given by

$$
\begin{aligned}
\dot{x}_f &= B_2 u_f \\
J_{\gamma_f} &= \tfrac{1}{2}\int_0^\infty \left(x_f^T Q_{22} x_f + u_f^T R u_f\right) d\tau
\end{aligned}
\tag{3.53}
$$

Solution of this fast LQ problem defined by Eq. (3.53) depends of the solution of

$$
Q_{22} = Z_f B_2 R^{-1} B_2^T Z_f \tag{3.54}
$$

which is given by

$$
Z_f = W^{-1}(W Q_{22} W)^{1/2} W^{-1} \tag{3.55}
$$

where $W = \left(B_2 R^{-1} B_2^T\right)^{1/2}$. The corresponding feedback controller for the fast subsystem is given by

$$
\begin{aligned}
u_f &= -B_2^T R^{-1} Z_f x_f \\
&= -K_{1f} x_f
\end{aligned}
\tag{3.56}
$$

Let $\gamma > \gamma_s$, the high gain composite controller is obtained by substituting x_f from Eq. (3.52) and $\bar{x}_2$ from Eqs. (3.43) and (3.45), which results in

$$
\begin{aligned}
u &= -\varepsilon^{-1}\left(K_{1f} K_{1s} x_1 + K_{1f} x_2\right) \\
&= -\left(C_{21} x_1 + C_{22} x_2\right)
\end{aligned}
\tag{3.57}
$$

Moreover, the composite control is near optimal in the sense that for all $\gamma > \gamma_s$, there exists an $\varepsilon_\gamma > 0$ such that for all $\varepsilon > \varepsilon_\gamma$, the disturbance attenuation is attained for the full order system. This is proven, as in [4], (using the Implicit Function Theorem), by showing that the elements of the solution to Eq. (3.37), Z_{11}, Z_{12}, and Z_{22}, each have an asymptotic expansion in ε. Using this result, and applying the composite control to the system in Eq. (3.7), it is shown that J_γ has a finite cost for $\gamma > \gamma_s$.

3.4 High Gain Autopilot Design and Simulation Results

In this section linear and nonlinear simulation results are presented which represent the application of near optimal high gain H_∞ theory to missile autopilot design. The resulting H_∞ angle of attack (AOA) command autopilot is also compared to the high gain LQ design.

The missile configuration chosen for this initial study uses reaction jet thrusters (RCVs) for control. The pitch plane autopilot design model (using Eq. (3.1)) is given by

$$x_0 = \left[\int e_\alpha \quad \alpha \quad q \quad T_{RCS}\right]^T \tag{3.58}$$

$$A_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Z_\alpha & Z_q & Z_{RCS} \\ 0 & M_\alpha & M_q & M_{RCS} \\ 0 & 0 & 0 & -1/\tau_{RCS} \end{bmatrix}; \quad B_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/\tau_{RCS} \end{bmatrix}; \quad D_0 = \begin{bmatrix} 0 \\ Z_w \\ M_w \\ 0 \end{bmatrix}$$

where the Z_i represent the partial derivatives $\partial\dot{\alpha}/\partial x_i\big|_{TRIM}$ evaluated at the trimmed flight condition, and the M_i represent the partial derivatives of $\partial\dot{q}/\partial x_i\big|_{TRIM}$. The state feedback control law is given by

$$T_{RCS_c} = -\frac{1}{\varepsilon^2} C_2 x_0 \tag{3.59}$$

The autopilot was designed at the following flight conditions:

$$\alpha = [0 \quad 10 \quad \cdots \quad 90] \quad (deg)$$
$$V = [500 \quad 1000] \quad (ft/s)$$
$$h = [10 \quad 30] \quad (Kft)$$

The performance weighting matrices were chosen so that the linear AOA step response had a rise time less than 400 milliseconds. The linear system AOA step response for the flight condition α = 40°, V = 1000 ft/s, with h = 10 Kft is shown in Figure 3.1.

The gain scheduled control law (gains in Eq. (3.59)) was simulated for a 180° turn to the rear hemisphere maneuver, assuming that the missile (main engine) was boosting at 5000 lbs, and that the disturbance w is a white noise process representing vertical wind gusts. The nonlinear planar simulation vertical velocity (w in body coordinates) responses for both the LQ and the H_∞ autopilots are shown in Figure 3.2, with and without wind effects. For illustrative purposes, the

wind gust magnitude was chosen to be larger than what is typically encountered in order to test the disturbance rejection capabilities of the autopilots.

Figure 3.3 shows the nonlinear AOA responses for both autopilots tracking the AOA command. Note that the H_∞ autopilot exhibits better command following in the absence of the disturbance. The command following of both autopilots is degraded as a result of the wind disturbance. However, the H_∞ autopilot does a better job at rejecting the wind disturbance.

The RCV control activity for this maneuver is shown in Figure 3.4. Notice that H_∞ autopilot exhibits more control activity than the LQ autopilot. The large spike in the H_∞ simulation RCV thrust response is the result of a significant increase in the magnitude of the disturbance input matrix D_1 between flight conditions, resulting in a large change in the controller gains (due to the gain scheduling). This problem could be addressed through a redesign of the H_∞ controller at the higher AOA flight condition with modified design requirements (same design requirements were used at low and high AOA in this study).

Figure 3.5 shows the downrange versus crossrange trajectories for both autopilots flying through the turbulence. The turn performance of the H_∞ autopilot is slightly better than the LQ autopilot.

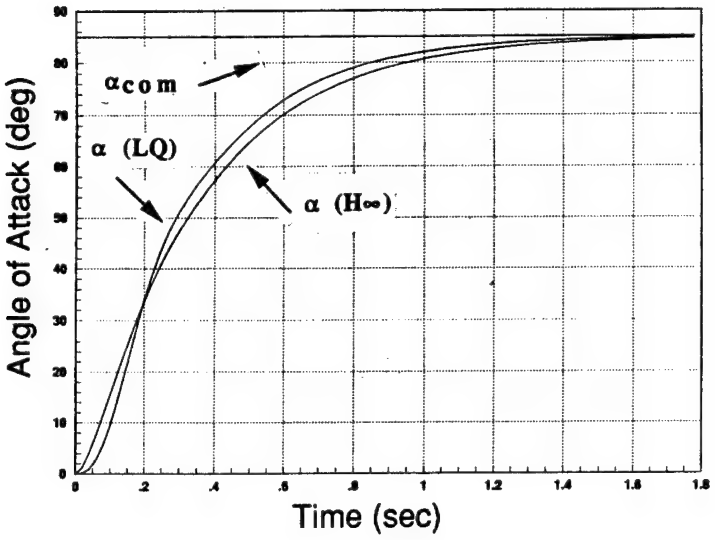
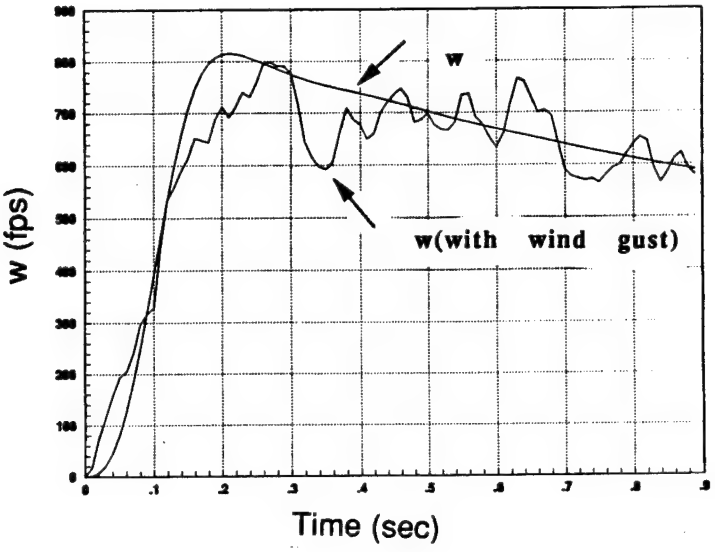
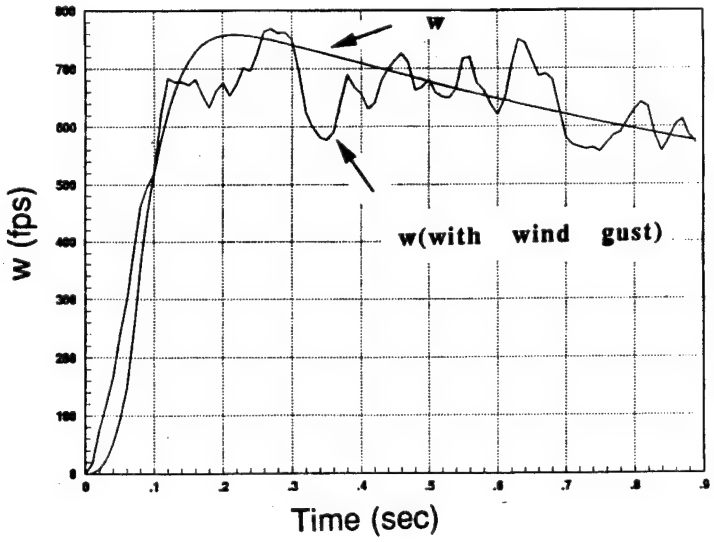


Figure 3.1 Linear angle of attack step responses.

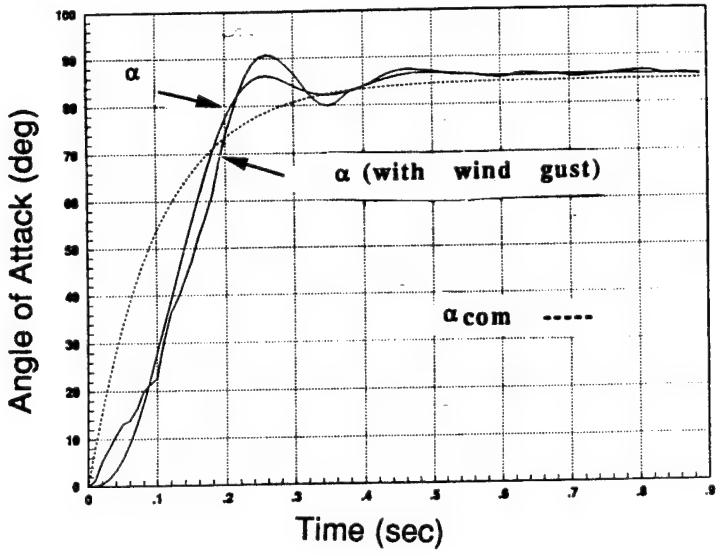


a) High Gain LQ Autopilot

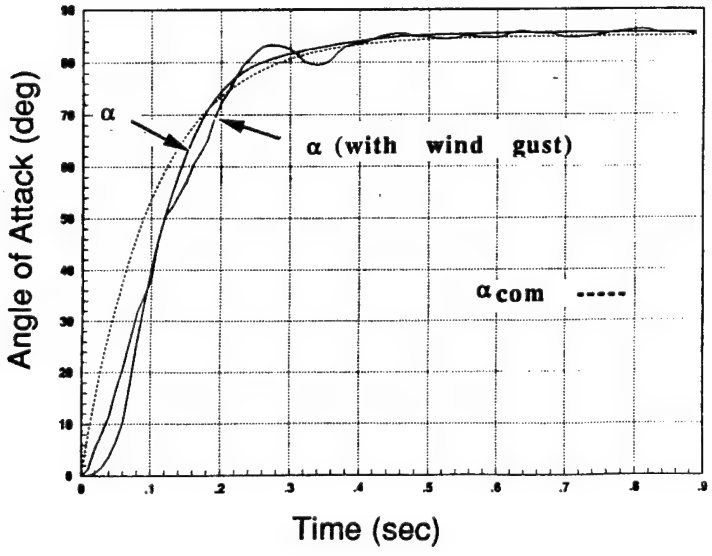


b) High Gain H_∞ Autopilot

Figure 3.2 Nonlinear planar simulation vertical velocity responses.

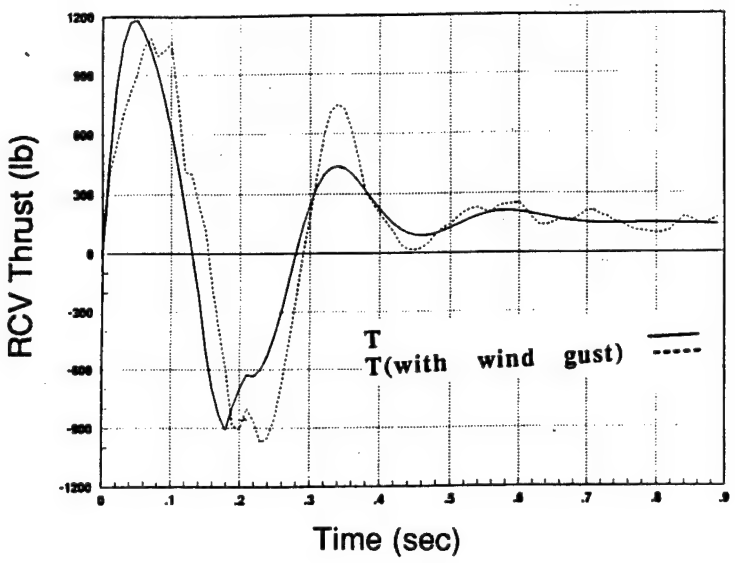


a) High Gain LQ Autopilot

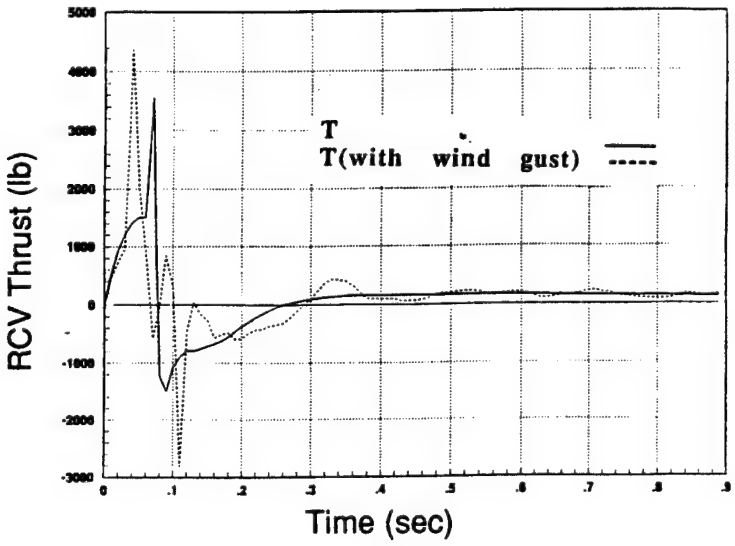


b) High Gain H_∞ Autopilot

Figure 3.3 Nonlinear planar simulation angle of attack time histories.



a) High Gain LQ Autopilot



b) High Gain H_∞ Autopilot

Figure 3.4 Nonlinear planar simulation reaction jet thrust responses.

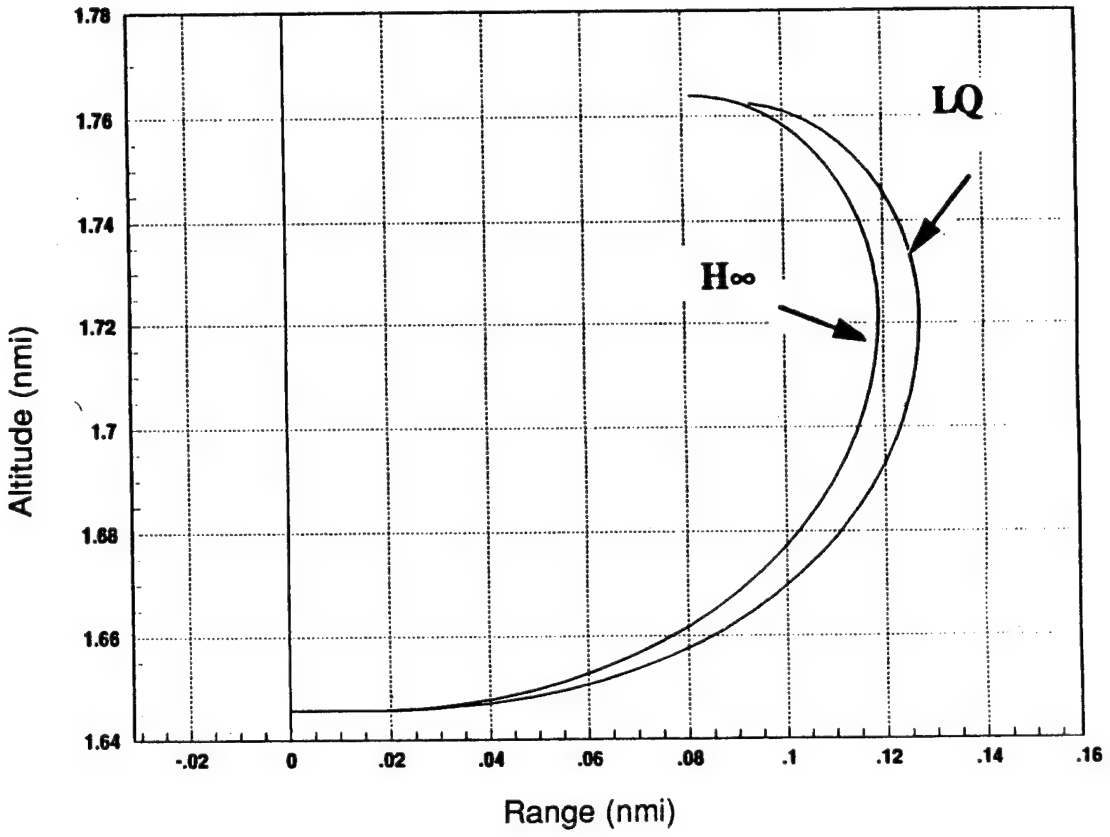


Figure 3.5 Nonlinear planar simulation downrange versus crossrange results.

3.5 Conclusion and Future Research

It has been shown that linear time invariant high gain feedback systems with additive disturbance can be represented as singularly perturbed systems, and therefore can be decomposed into slow and fast subsystems for control law design. Assuming that the disturbance is a linear function of the slow state, it was shown that the fast subsystem dynamics are unaffected by the disturbance. However, the stability of the slow subsystem is dependent upon the disturbance feedback gain. The solution to the near optimal high gain H_∞ state feedback problem was then shown to be equivalent to the solution of a slow H_∞ state feedback problem and a fast LQ state feedback problem. For sufficiently large γ, the conditions for the existence of solutions are shown to be equivalent to the existence conditions for the high gain LQ problem. Moreover, the optimality of H_∞ composite control system with respect to the H_∞ performance measure is equivalent to the optimality of the LQ composite control with respect to the LQ performance measure, for ε sufficiently small. Finally, nonlinear simulation results showed that command tracking performance of the high gain H_∞ autopilot was comparable to the high gain LQ autopilot. However, for the 180° maneuver in the presence of turbulence, the H_∞ autopilot exhibits slightly better disturbance rejection properties.

Future research needs to be directed at the main issues in applying VSC sliding mode control law design and analysis to missile autopilot problems, that is sliding mode existence, asymptotic stability of the sliding mode, and reachability to the sliding mode. Conditions for evaluating these features are well known for linear time invariant (LTI) systems [4], and readily support control law design using LTI system models with gain scheduling.

Modeling a missile's dynamics using LTI system models naturally leads to a gain scheduled control law. This is typical of how industry designs missile control laws. Applying VSC to these LTI models, as demonstrated in the work performed here, leads to a gain scheduled sliding surface, with the gains interpolated between design points. Verification of the design relies on proof by nonlinear simulation.

Another approach to the VSC missile autopilot design problem is to design a nonlinear control law. This requires designing a nonlinear sliding surface, $s(x)$, for the nonlinear system

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ u &= u_{max} sgn(s(x)) \end{aligned} \tag{3.60}$$

Asymptotic stability of the above system is based upon the following fact: If the sliding mode is asymptotically stable, then the closed loop system is asymptotically stable if and only if the sliding mode is reachable. The goal is then to determine $s(x)$ to insure both asymptotic stability of the sliding mode and reachability to the sliding mode. An approach to address nonlinear sliding mode stability is outlined below. Methods to evaluate reachability represent new research.

The condition for sliding mode stability is based upon passivity and detectability of the system as follows. If the system is passive with a positive definite C^1 storage function $V(x)$, then the sliding mode is asymptotically stable provided that it is detectable. By definition, the system given by

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= s(x) \\ u &= -\phi(s(x)) \end{aligned} \tag{3.61}$$

is passive if there exists a function $V: R^n \rightarrow R$, with $V(x) \geq 0$, $V(0) = 0$, and $V(x) \in C^1$, such that

$$V(x(t)) - V(x(t_0)) \leq \int_{t_0}^{t} y^T(s)u(s)ds \tag{3.60}$$

A condition for determining passivity of a given system is the Kalman-Yakubovitch-Popov (KYP) Lemma which. Let $s(x) = \left(L_g V\right)^T$ where $V(x) \geq 0$, $V(0) = 0$, and $V(x) \in C^1$. Then, if $L_f V \leq 0$, then the system described by Eq. (3.59) is passive.

Some of the issues involved with using this approach are verifying the existence of $V(x)$, determining $V(x)$, and determining u as a function of $s(x)$. Nonlinear optimal control theory could potentially be used to address be these issues.

Detectability of the system in Eq. (3.59) is evaluated by examining the set

$$S = \left\{ \bar{x} : L_f^m L_{[f,[f,\cdots[f,g]\,]} V(\bar{x}) = 0 \right\}$$

where m is the rank of $L_g V$. If $S = \{0\}$, then the system in Eq. (3.61) is detectable.

One of the major problems in performing nonlinear control law design is developing a representative design model. The aerodynamic coefficients are typically given in a table lookup form as a function of Mach, control effector position, body rates, and wind angles. Fitting a

model to this data can require an extensive amount of work. In addition, there are significant uncertainties in the wind tunnel data due the measurement instrumentation. These uncertainties and high alpha phenomena such as asymmetric vortex shedding can further complicate the modeling process. In addition, nonlinear control law design techniques are more complicated and the performance payoff (over gain scheduled designs based upon LTI models) is not clear. On the other hand, nonlinear analysis techniques are always required and may help quantify the performance advantages when using nonlinear control techniques. To compare the VSC gain scheduled control law to a VSC nonlinear control law, further research is needed to develop reaching conditions for the nonlinear system, and in combining these conditions with the sliding mode asymptotic stability conditions obtained via the KYP Lemma (assuming detectability).

3.6 Chapter 3 References

[1]. A. Jameson and R. O'Malley, Jr., "Cheap Control of the Time Invariant Regulator", Appl. Math. Opt., vol. 1, no 4, pp. 337-354, 1975.

[2]. D. Young, P. Kokotovic, and V. Utkin, "A Singular Perturbation Analysis of High-Gain Feedback Systems", IEEE Trans. Automat. Contr., vol. AC-22, No. 6 Dec. 1977.

[3]. J. Chow and P. Kokotovic, "A Decomposition of Near-Optimum Regulators For Systems With Slow and Fast Modes", IEEE Trans. Automat. Contr., vol. AC-21, pp. 701-705 Oct. 1976.

[4]. Z. Pan and T. Basar, "H_∞ Optimal Control for Singularly Perturbed Systems", Part 1, Automatica, Nov. 1992.

[5]. P. Kokotovic and A. Haddad, "Controllability and Time-Optimal Control of Systems With Slow and Fast Modes", IEEE Trans. Automat. Contr., vol. AC-20, pp. 111-113, Feb. 1975.

[6]. T. Basar and P. Bernhard, *H_∞ Optimal Control and Related Minimax Design Problems, A Dynamic Game Approach*, Birkhauser, 1991.

[7]. Weil, R. D., and K. A. Wise, "Blended Aero & Reaction Jet Missile Autopilot Design Using VSS Techniques," Proc. of the 30th IEEE CDC, Brighton, UK, Dec. 11-13, 1991.

4 Nonconservative Robustness Tests For Mixed Uncertainties

Robustness analysis is important because of the uncertainty present in engineering design models of physical systems. This uncertainty comes from many sources such as parameter value uncertainties, neglected and/or mismodeled dynamics, time delays, and neglected nonlinearities. These modeling errors fall into two broad categories: *parametric* (real) and *dynamic* (complex) uncertainty. A more realistic description of these errors is one that includes both categories, which is referred to as *mixed* modeling uncertainty. Feedback control is used to achieve performance in the presence of such uncertainties.

Contrary to the 1970's where control research dealt with systems with known mathematical models, control engineers today must design for performance and understand the impact of uncertainty. The pioneering work of Zames, Doyle, Stein, and Safonov has led to a framework in which robustness analysis and synthesis, for restricted types of uncertainties, is now possible.

There are many techniques available for robustness analysis. These stability tests focus on either parametric or dynamic uncertainty, and can be categorized as polynomial tests, Lyapunov tests, zero exclusion tests, and singular value based tests. When analyzing a missile flight control system, robustness to both parametric *and* dynamic uncertainty is of critical importance.

The research presented here in robustness analysis is motivated by a general industry need to guarantee stability and performance robustness for control systems with mixed uncertainty, and secondly, by a need to reduce the costs associated with designing these control systems. Missile flight control systems use gain schedules designed from linearized models to compensate for highly nonlinear aerodynamic characteristics, undesirable gyroscopic coupling, and the large range of flight conditions encountered over the flight envelope. Automation of this task using robustness theory is shown in Figure 4.1.

Using the approach illustrated in Figure 4.1, feedback gains are calculated at each design point using controller synthesis software. Robustness analysis software is used to determine the next design point in the flight envelope, guaranteeing uniform stability margins as well as performance between design points. This loop is repeated until the flight envelope is covered by the gain schedule. Nonconservative evaluation of the flight control system robustness to mixed uncertainty is required in order to automate the process.

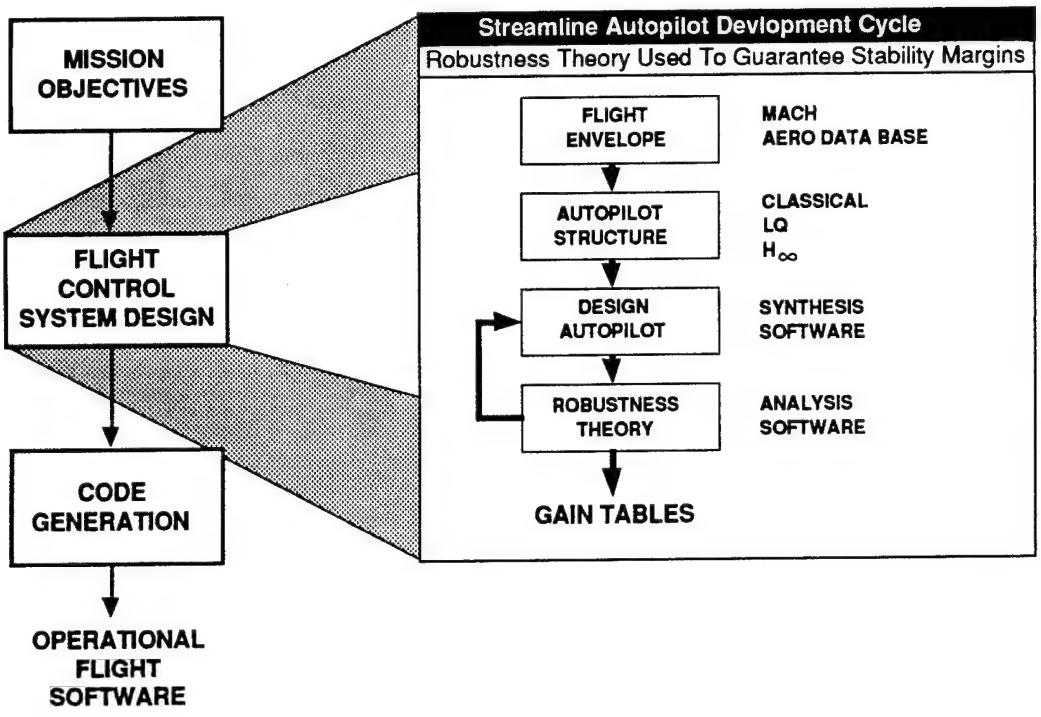


Figure 4.1 MDC automated flight control system design.

Results using this process have been published in Wise [1] using the structured singular value μ (using complex μ for mixed uncertainty) robustness analysis, and are shown in Figure 4.2.. This approach led to large gain tables in which the numerical values of the gains changed very little between design points (due to the conservatism of complex μ). The missile's aerodynamics become unstable at 13° AOA. As shown in the figure, the gain tables became very dense in this region with very little change in magnitude.

Stability robustness to real parameter uncertainties have been analyzed using the deGaston-Safonov real multiloop stability margin [2], with significant improvements in the algorithm made by Sideris [3] in removing the frequency search. For dynamic uncertainties the most popular method of analysis is Doyle's structured singular value [4-6]. This pioneering work has been extended to address mixed (real and complex) uncertainty analysis problems in Young [7].

The uncertainties in a typical feedback control system may arise from real parameter variations, neglected/mismodeled dynamics, or combinations of both (mixed uncertainty). The stability analysis model is shown in Figure 4.3. The uncertainties in the system are isolated and placed into a diagonal matrix Δ. The transfer matrix M describes nominal system characteristics which have been stabilized by a compensator. Thus, for $\Delta = 0$, the system is stable. Let

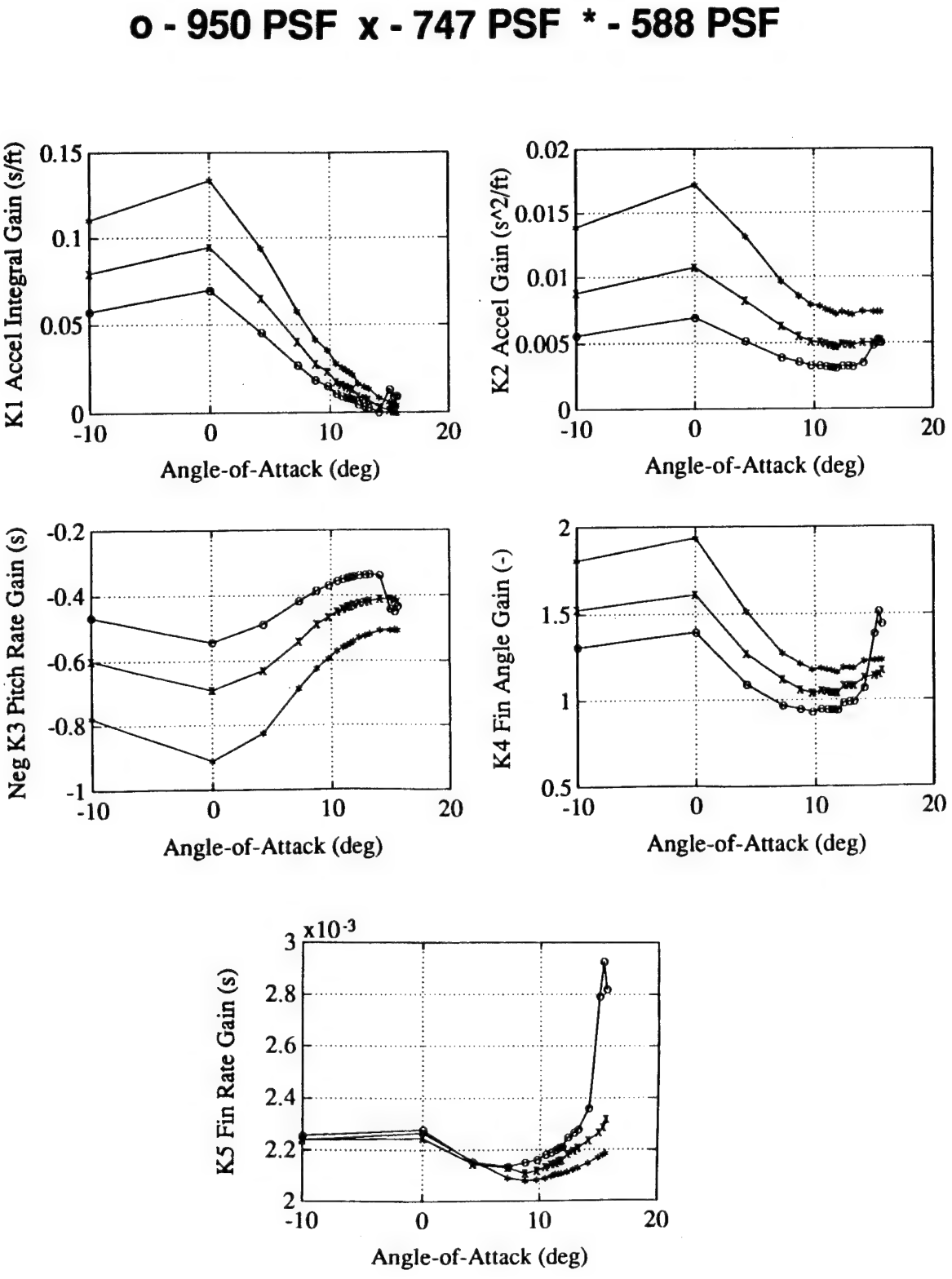


Figure 4.2 Pitch autopilot feedback gains.

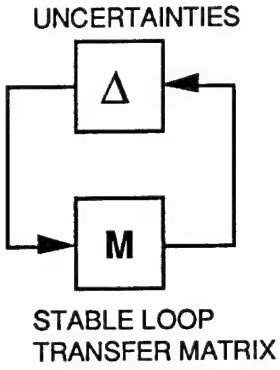


Figure 4.3 ΔM uncertainty analysis model.

$$\Delta = diag[\delta_1 \quad \cdots \quad \delta_n]$$
$$M(j\omega) \in C^{n \times n} \tag{4.1}$$

Stability of the system described by Figure 4.3 is implied by the $det[I - \Delta M] \neq 0$.

The algorithm presented here offers the analyst an alternative approach (other than μ) for analyzing stability robustness to mixed uncertainties. The idea is to determine the smallest set of uncertainties that makes the return difference matrix singular, with the search performed over the parameter space that models the uncertainties. Although the algorithms require 2^n operations at least once (n = number of uncertainties), the approach is still reasonable (for small to medium sized problems) for many engineering problems of interest to the aerospace industry.

The stability robustness analysis problem is solved by first forming an analysis model (ΔM) in which the uncertainties are isolated into the Δ matrix. Next, a variation polynomial $a(\delta)$ is formed by expanding the determinant of the return difference matrix for this analysis model ($det[I - \Delta M] = a(\delta)$). The robustness test determines what uncertainties Δ make the return difference matrix singular (i.e. $det[I - \Delta M] = 0$) by computing the zeros of the variation polynomial $a(\delta)$. The zeros of $a(\delta)$ are found by using a conjugate gradient algorithm minimizing the magnitude squared of the polynomial $a(\delta)$, combined with a simulated annealing algorithm for starting the conjugate gradient optimization.

4.1 The Variation Polynomial

The variation polynomial $a(\delta)$ is formed by the determinant expansion of $det[I - \Delta M]$, where the diagonal matrix Δ models mixed uncertainties and the nominal matrix M is stable. The zeros of the variation polynomial $a(\delta)$ determine when the mixed uncertainties destabilize the system. Our objective is to form the polynomial $a(\delta)$ at each frequency and solve, by optimization, for the destabilizing uncertainties.

The following three steps are used to form the ΔM analysis model from which we compute the variation polynomial $a(\delta)$:

i) build a signal flow graph (or block diagram) model with scalar uncertainties.

ii) form the ΔM analysis model using appropriate software isolating the uncertainties into a diagonal matrix Δ (this step can be accomplished using MATRIXx , CTRL-C, Matlab).

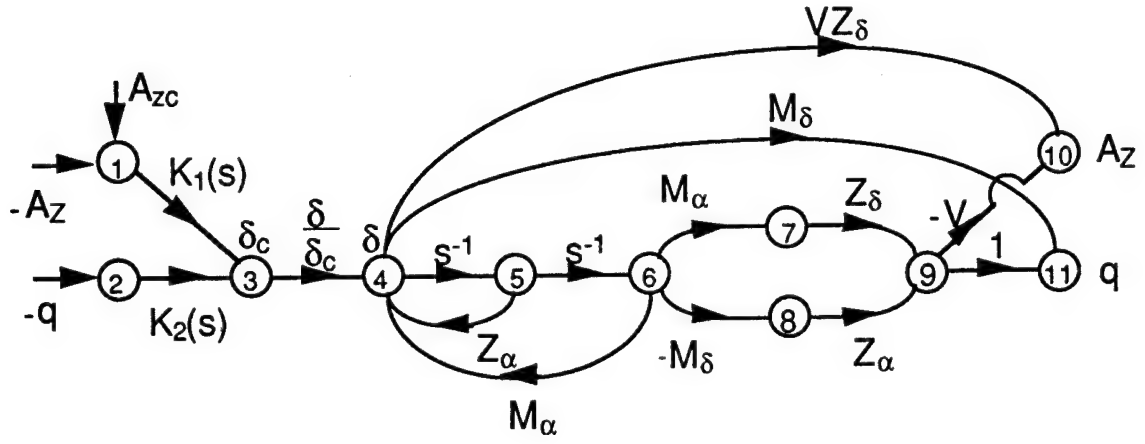
iii) compute the variation polynomial $a(\delta)$ coefficients from $a(\delta) = det[I - \Delta M]$.

Step i)

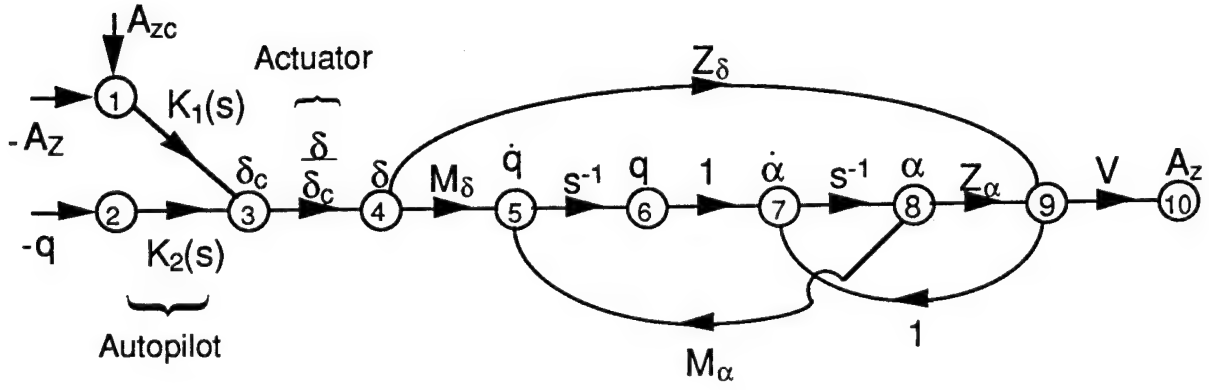
The ΔM robustness analysis model is formed by introducing scalar uncertainty models into a signal flow graph model of the control system, and manipulating the signal flow graph. All parametric (real) and dynamic (complex) uncertainties are modeled using scalars, and the resulting analysis model *always* has a diagonal matrix Δ.

Figure 4.4 illustrates this point by showing two signal flows graph models of the longitudinal flight control system of a bank-to-turn missile. Figure 4.4a was created from transfer functions whereas Figure 4.4b was created from the state equations.

Parametric uncertainties in real parameters p_j can be modeled using a multiplicative uncertainty model $p_j = \bar{p}_j(1 + \delta p_j)$. Dynamic (complex) uncertainties can be modeled using complex scalars c_i and also can use a multiplicative uncertainty model $c_i = \bar{c}_i(1 + \delta c_i)$. (Additive uncertainties are also modeled in this framework.) These parametric and dynamic uncertainties are modeled using signal flow graph branches and are inserted into the system signal flow graph, as shown below.



a) Signal flow graph from transfer functions.



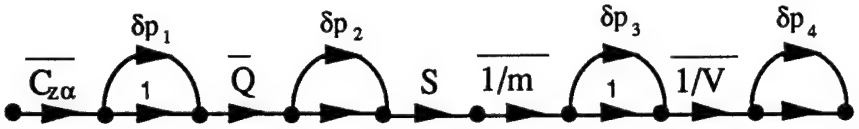
b) Signal flow graph from state equations.

Figure 4.4 Signal flow graph models for a pitch plane dynamics and autopilot.

Step ii)

The matrix M is computed by breaking the signal flow branches at each variation and forming input/output nodes as shown above. The elements of M are the transfer functions between the input and output nodes of the signal flow graph. The motivation for using signal flow graph branch models is that multi-parameter coefficients can be expanded using branch gain models, and variations can be introduced into the individual parameters. For example, the missile stability derivative Z_α (body lift due to angle-of-attack α) can be expanded as

$$Z_\alpha = \frac{C_{z\alpha} QS}{mV}$$



which allows for simultaneous variations in the nondimensional stability derivative $C_{z\alpha}$, dynamic pressure $\overline{Q}$, mass m, and velocity V (in a real parameter uncertainty analysis problem). This signal flow graph modeling technique removes any restrictions on how the variations can enter into the model. A FORTRAN implementation of our signal flow graph model decomposition uses Cramer's rule (Klein [8], Wise [9,10]) to form the transfer function matrix $M(j\omega)$. We have also implemented this approach in Matlab, CTRL-C, and Matrix-X.

The ΔM analysis model that results from our signal flow graph decomposition is highly structured, and can be used in analyzing specific uncertainty models, when norm bounded uncertainty representations are available, or to guarantee uniform stability margins over a range of real parameter variations. The mixed uncertainties δp_i and δc_i model parameter variations that appear linearly-fractionally in the system model. Our research also applies to systems whose parameters do not appear linearly-fractionally in the plant model, such as the parameters ϕ_i in complex exponentials $e^{-j\phi_i}$ (time delays added at the plant inputs and outputs).

Step iii)

The expansion of the $det[I - \Delta M]$ forms an affine polynomial in the parameter variations δp_i and δc_i. Consider, for example, $S_1 = 2$ real parameter variations and $S_2 = 1$ complex variations, with $n = S_1 + S_2 = 3$ the dimension of Δ. The variation polynomial has 2^n coefficients $a_k \in C$ and is formed as follows for this example:

$$\Delta = diag[\delta p_1 \;\; \delta p_2 \;\; \delta c_1] \tag{4.2}$$

$$\begin{aligned} a(\delta) &= det[I - \Delta M] \\ &= a_0 + a_1\delta p_1 + a_2\delta p_2 + a_3\delta c_1 + a_4\delta p_1\delta p_2 + a_5\delta p_1\delta c_1 + a_6\delta p_2\delta c_{11} + a_7\delta p_1\delta p_2\delta c_1 \end{aligned}$$

This can be written as the inner product of two vectors by factoring out the coefficients a_k into the vector a and placing the parameter uncertainties into the vector δ as follows:

$$\begin{aligned} det[I-\Delta M] &= [a_0 \quad \cdots \quad a_7][1 \quad \delta p_1 \quad \delta p_2 \quad \delta c_1 \quad \cdots \quad \delta p_1 \delta p_2 \delta c_1]^T \\ &= a^T \delta \\ &= a(\delta) \end{aligned} \tag{4.3}$$

The $2^n \times 1$ vector a is computed at each frequency by mapping the vertices of a particular parameter space hypercube into the complex plane using the $det[I-\Delta M]$. By stacking each mapped vertex into a vector v, $v \in C^{2^n}$, we can relate v to the coefficient vector using

$$v = Pa \tag{4.4}$$

where each row of the matrix P is δ evaluated at a vertex of the hypercube. By a special choice of the parameter space hypercube we can design the matrix P to be orthogonal (this applies to both real and complex uncertainties). This lets us invert the $2^n \times 2^n$ matrix by taking its transpose, allowing us to easily solve for the coefficient vector a in Eq. (4.4).

$$a = P^T v \tag{4.5}$$

For real parameter variation problems, we have used Eq (4.4) in computing the deGaston-Safonov real multivariable stability margin. Some very interesting observations resulted, and are presented in the following numerical results subsection on parametric uncertainty. Faced with an analysis problem in which the uncertainties did not appear linearly-fractionally in the model (time delays), we extended this robustness analysis method to include complex parameter variations. These results follow in the numerical results subsection on dynamic uncertainty.

4.2 Computing the Zeros of the Variation Polynomial

In the general case of n parameter variations (complex parameter variations or mixed real and complex parameter variations) optimization techniques are employed to compute the zeros of the variation polynomial $a(\delta)$. The objective function used is the magnitude squared of the variation polynomial. The zeros of $a(\delta)$ are found by using a conjugate gradient algorithm minimizing the magnitude squared of the polynomial $a(\delta)$, combined with a simulated annealing algorithm for starting the conjugate gradient optimization. The analysis goal is to find the smallest set of parameter uncertainties that make $a(\delta) = 0$.

There are many techniques available to find the minimum value of an objective function. Of all these methods the conjugate gradient method [11-14] is the simplest.

The problem of minimization can be visualized as a problem in hill climbing [11]. The bottom of the valley can be found by starting at some initial point and climbing in the downward direction until a minimum point is reached. The climbing procedure is efficient if the direction of climbing is the direction of steepest descent. A detailed algorithm for the steepest descent method is given in [11]. The steepest descent method can be modified to take the advantage of mutually conjugate directions of descent [12]. This reduces the convergence difficulties of the steepest descent method presented in [11]. Most of the conjugate gradient algorithms presented in [11-13] deal with objective functions which are quadratic. Conjugate gradient algorithms used to optimize non-quadratic functions are presented in [14].

The zeros of $a(\delta)$ can be found by using a conjugate gradient algorithm for non-quadratic functions as presented in [14]. Since $a(\delta)$ is complex, the objective function to be minimized is written as follows

$$F = \left(a^T \delta\right)\left(a^T \delta\right)^* \tag{4.6}$$

Where $a(\delta)$ is the complex variation polynomial and $(\bullet)^*$ is it's complex conjugate. When F becomes zero $a(\delta)$ is also zero.

The objective function F presented in Eq. (4.6) is one equation in n unknowns. Also, there is more than one set of parameters that make the polynomial zero. The set of the smallest uncertainties that make the polynomial zero is to be found. This type of combinatorial optimization problem can be solved using an approach called "simulated annealing". Simulated annealing is analogous to the physical process of annealing of solids [15].

Annealing is a process in which a solid is heated in a heat bath until the solid melts. Then the temperature of the heat bath is reduced gradually until the particles arrange themselves in the ground state of the solid. At each temperature value T, the solid is allowed to reach thermal equilibrium. This thermal equilibrium is characterized by the Boltzmann distribution which is stated as below:

$$P_T\{X = i\} = \frac{1}{Z(T)} exp\left(\frac{-E_i}{\kappa_B T}\right) \tag{4.7}$$

Where X denotes the current state of the solid and $Z(T)$ denotes the partition function [15], which is defined as follows

$$Z(T) = \sum_{j=1}^{n} exp\left(\frac{-E_i}{\kappa_B T}\right) \tag{4.8}$$

where E_i is the energy, κ_B is the Boltzmann's constant, and T is the temperature of the heat bath.

The analogy between the simulated annealing and the physical annealing process is that optimal solutions of a combinatorial optimization problem are analogous to the states in a physical annealing process. Similarly, the cost of an optimal solution is analogous to the energy of each state.

Software Implementation

The software tool developed under this research (called ROBUSTC) builds upon the FORTRAN program called ROBUSTR published in Wise [9, 10]. ROBUSTR was developed to calculate the deGaston-Safonov [2] real multiloop stability margin. It is applicable only to LTI control systems with real parameter variations.

In implementing the calculation of the real multiloop stability margin the variation polynomial $a(\delta)$ was formed [16, 17]. This polynomial is the expansion of the determinant of the return difference matrix $I - \Delta M$ for the system shown in Figure 4.3. It can be written as an inner product of two vectors, and can be used to replace the matrix determinant calculations required to map the parameter space hypercube into the Nyquist plane when computing the real margin. This approach led to a significant reduction in the CPU time required to compute the real margin using the deGaston-Safonov algorithm.

Figure 4.5 illustrates the software development process that was used to develop ROBUSTC. Prior to testing ROBUSTC, the individual subroutines underwent unit testing. The conjugate gradient subroutine used in ROBUSTC (called FRPRMN) went through an extensive testing procedure comparing it with another well documented conjugate gradient subroutine called VMCON (used at MDC). An analytical expression for the gradient is used in ROBUSTC. The analytical gradient was compared with numerical approximations of the gradient, for small changes in the parameters, with both methods giving identical results. After these individual subroutines were tested, the overall program was assembled.

Figures 4.6 through 4.8 show a flow charts for the FORTRAN program ROBUSTC. Any analysis model can be input to ROBUSTC by first forming a state space model that describes the M matrix in Figure 4.3, using either Matlab, CTRL-C, or MATRIXx. The analysis model should be validated prior to building the state space model for ROBUSTC. A Matlab procedure file was

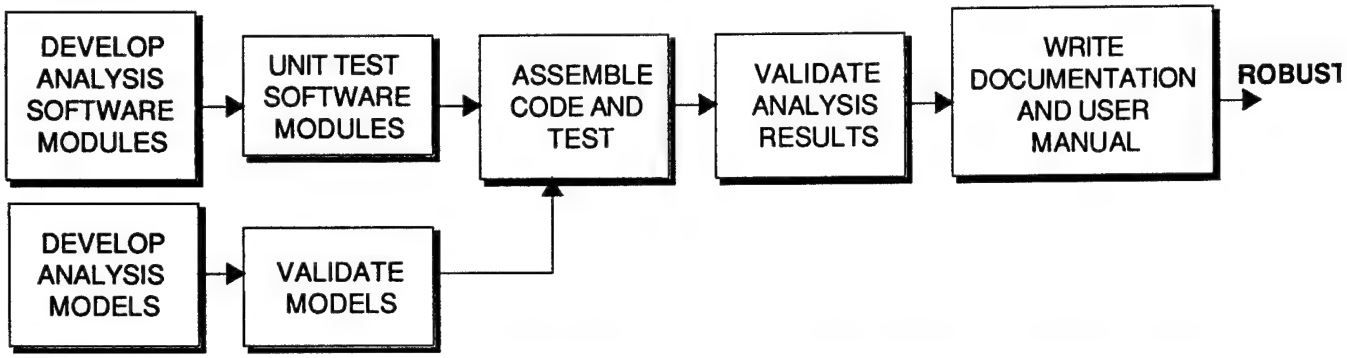


Figure 4.5 ROBUSTC analysis software development.

written which takes any M-matrix quadruple, (A_m, B_m, C_m, D_m), and writes the FORTRAN code that inputs the model into ROBUSTC. This code is then compiled and linked to the ROBUSTC code.

Simulated annealing finds the global minimum by jumping around in the parameter space (due to the high temperatures) and evaluating the cost at each jump. The smallest cost and parameters associated with that cost are stored for retrieval. The algorithm stops when the process has sufficiently cooled, as specified by the user.

In our application of simulated annealing, the annealing algorithm is used to start a conjugate gradient algorithm that minimizes the variation polynomial (F in Eq. (4.6)). The objective function is not a convex function, so local minima can result, depending upon where the algorithm is started. The annealing algorithm, by jumping all over the parameter space for the uncertainties, and starting the conjugate gradient algorithm at each jump, finds the smallest set of destabilizing parameter uncertainties.

An algorithm to implement the simulated annealing is presented in [15]. A flowchart for the implementation of this algorithm for our problem is given in Figures 4.7 and 4.8, and is explained as follows. The function FRPRMN uses a conjugate gradient algorithm to minimize the variation polynomial objective function F. The input argument to this function is some starting value DELTA where the conjugate gradient algorithm is going to start. This function returns two arguments; 1) the local optimal solution of F which is denoted as DELTA*; and 2) the optimal

value of the function F which is denoted as FVAL*. To find the smallest set of destabilizing uncertainties, the 2-norm of the optimal solution DELTA* is calculated, which is denoted as COST;

The function logsched returns a zero or a one depending on the input arguments. This output of this function is used to adjust the temperature, cooling the process when the cost is reduced.. The flow chart for logsched is given in Figure 4.8. The function rand generates a vector of random numbers.

The development and validation of the analysis models prior to using ROBUSTC is a key step in process. Experience has shown that it is easy to make modeling mistakes when analyzing complicated flight control problems containing a large number of uncertain parameters.

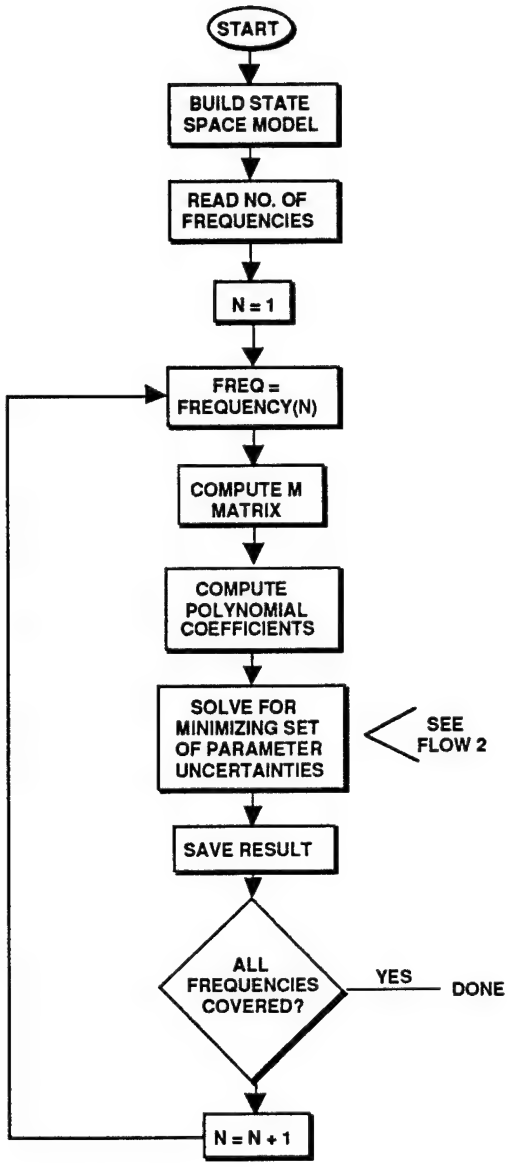


Figure 4.6 Flow chart of ROBUSTC.

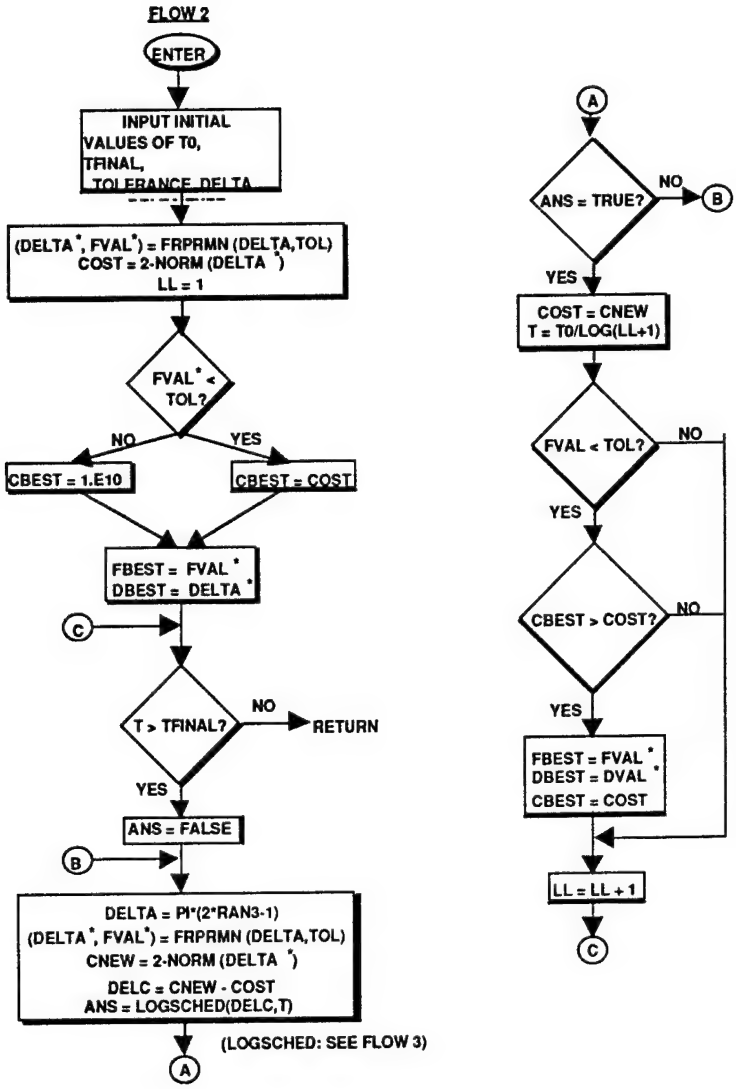


Figure 4.7 ROBUSTC simulated annealing implementation.

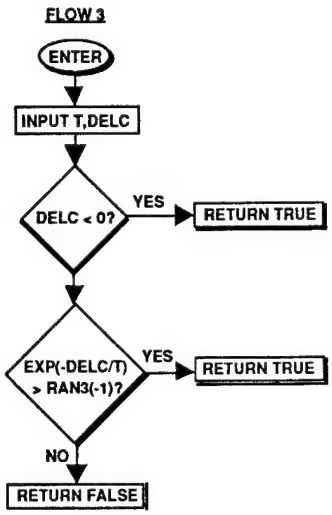


Figure 4.8 Logsched flow chart.

4.3 Numerical Results

Parametric Uncertainty (Real)

For problems with only real parametric uncertainty the robustness analysis problem can be solved by the deGaston-Safonov real multivariable stability margin [2]. We have implemented Eq (4.3) in FORTRAN software for computing the real margin k_m. To start, the algorithm requires mapping the 2^n vertices of the parameter space hypercube into the Nyquist plane at a specific frequency. Each vertex of the cube defines a Δ and is mapped by the $det[I - k\Delta M]$. This requires evaluating the determinant of a $n \times n$ matrix 2^n times. Once this is complete, Eq. (4.5) is used to solve for the variation polynomial coefficients. During the search for the largest k, Eq. (4.3) replaces the determinant calculation with a vector inner product. In every application of this approach, we have noticed that a majority of the coefficients a_k were zero, and remain zero for all frequency. For the remaining frequencies (during the frequency sweep), only the nonzero variation polynomial coefficients are calculated. This greatly reduces the size of the variation polynomial that must be considered, and the size of the vectors used to compute its magnitude.

We have incorporated the variation polynomial approach along with a polynomial-time convex hull algorithm into a generic real parameter variation analysis program called ROBUSTR (see [9,10] for details on this software). Table 4.1 lists CPU times using a Vaxstation 3200 workstation for several missile autopilot analysis problems. In an analysis of a roll-yaw missile autopilot with $n = 9$ uncertain real parameters, only 32 of the 512 variation polynomial coefficients were nonzero. This reduced Eq (4.3) to an inner product of two 32×1 vectors, greatly reducing the computational burden. This important observation, an implementation, makes the real margin algorithm implementable for *n*-large problems.

ROBUSTC has been applied to several test problems (generally low order) where analytical results can be computed. Figure 4.9 shows a block diagram and signal flow graph model for a real parameter variation analysis problem. These results can also be calculated using ROBUSTR which implements the deGaston-Safonov real margin algorithm. The closed loop characteristic equation for the system shown in Figure 4.9 is given by

$$s^3 + 10s^2 + 150(1 + \delta k_1)s + 960(1 + \delta k_2) = 0 \qquad (4.9)$$

Substituting $s = j\omega$ with $\omega = 1$ rad/s results in

$$950 + 960\delta k_2 + j(149 + 150\delta k_1) = 0 \qquad (4.10)$$

ROBUSTR CPU USAGE

AUTOPILOT	UNCERTAIN PARAMETERS (n)	MODEL INPUTS (n_p)	NODE EQUATIONS (n_a)	CPU TIME TO FORM MODEL (SEC)	ANALYSIS CPU TIME (SEC)	New Algorithm
PITCH						
PITCH-RATE	4	4	13	0.6	43	
COMAND	4	7	23	2.9	161	
PITCH	4	4	18	1.6	44	11
ACCELERATION	4	6	21	4.3	110	
COMMAND	4	8	28	10.5	229	
ROLL-YAW	4	4	40	57.5	38	7
ROLL-RATE	5	5	40	78.0	164 (0.04 HRS)	22
COMMAND	6	6	40	96.5	289 (0.08 HRS)	33
	7	7	40	117.5	1033 (0.28 HRS)	157 (0.04 HRS)
	8	8	40	136.2	3093 (0.86 HRS)	458 (0.13 HRS)
	9	9	40	156.4	13683 (3.80 HRS)	2039 (0.57 HRS)

VAX STATION 3200

Table 4.1 ROBUSTR CPU usage.

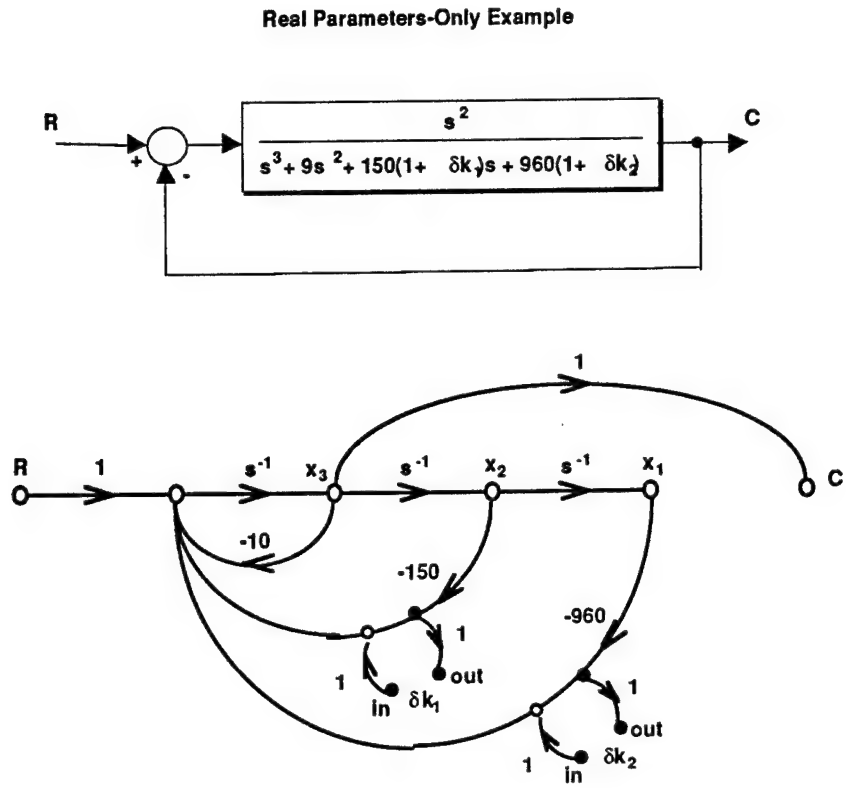


Figure 4.9 Real uncertainty 2-parameter test problem.

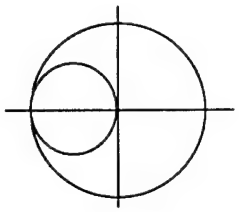
The parameters δk_1 and δk_2 that make the real and imaginary terms zero are:

$$\delta k_1 = -0.993333$$
$$\delta k_2 = -0.989583 \qquad (4.11)$$

These same results are computed by ROBUSTC.

Dynamic Uncertainty (Complex)

We have also used our signal flow graph modeling technique to develop analysis models for parameter variations that do not appear linearly-fractionally in the model, such as parameters $\phi_i \in [-\pi, \pi]$ that appear in $e^{-j\phi_i}$. For a time delay uncertainty, the multiplicative uncertainty model is $e^{-j\phi_i} = 1 + \delta_i$. Solving for the uncertainty δ_i yields $\delta_i = 1 - e^{-j\phi_i}$. The SSV μ produces conservative results when applied to this problem. The multiplicative uncertainty $e^{-j\phi_i} - 1$ (disk of radius one centered at -1) is conservatively covered by a disk (disk of radius 2), as shown below.



The analysis problem that motivated this applied research was an automatic landing control system for an Unmanned Air Vehicle (UAV). The current production UAV is landed by an operator using a double-joystick (throttle plus control surfaces), which requires extensive training. In addition, some mission scenarios include multiple UAV's which increases the operator work load. An AUTOLAND system would reduce training costs, improve the landing success rate, and reduce operator work load. In the AUTOLAND control system, telemetry data from the vehicle along with ground radar data is processed in a ground control station and the flight control signals are sent back to the UAV. The analysis question was to determine if the transportation delays destabilize the UAV, and how large could they be?

In addition to this motivation, plant-input and output stability margins can be computed using the gain and phase variation model $k_i e^{-j\phi_i}$ inserted in each input and output channel. Gain margins would be evaluated with the ϕ_i set equal to zero, and phase margins would be evaluated with the gain variations k_i set to unity. However, the ϕ_i do not appear linearly-fractionally in the model.

Following these motivations, consider the missile pitch autopilot phase variation analysis problem shown in Figure 4.10. Using a multiplicative uncertainty model for each $e^{-j\phi_i}$, as shown in Figure 4.10, our approach yields the following variation polynomial:

$$a(\delta) = a_0 + a_4 e^{-j(\phi_1+\phi_2)} + a_5 e^{-j(\phi_1+\phi_3)} \tag{4.12}$$

where the five coefficients in this variation polynomial that are not listed were identically zero for all frequency. The zeros of $a(\delta)$ occur at the ϕ_i's that destabilize the system. Figure 4.11 shows the frequency response of the coefficients a_k.

For $\delta_i = e^{-j\phi_i}$, the sum of the coefficients a_k in Eq. (4.12) always equal unity. At high frequencies, the variation polynomial cannot be made equal to zero by adjusting the parameters ϕ_1, ϕ_2, or ϕ_3. This is evident from Figure 4.11 as the magnitude of *a4* and *a5* go to zero.

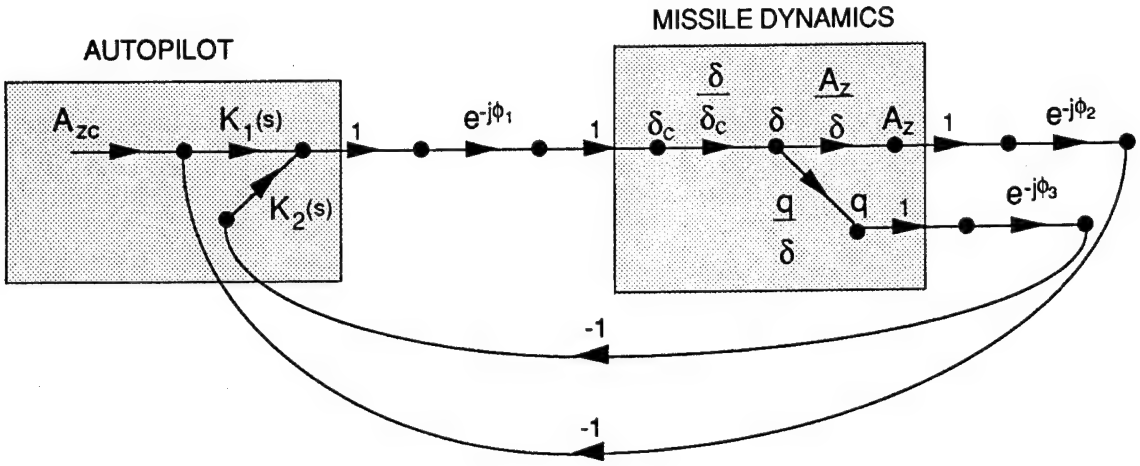


Figure 4.10 Pitch autopilot phase variation analysis model.

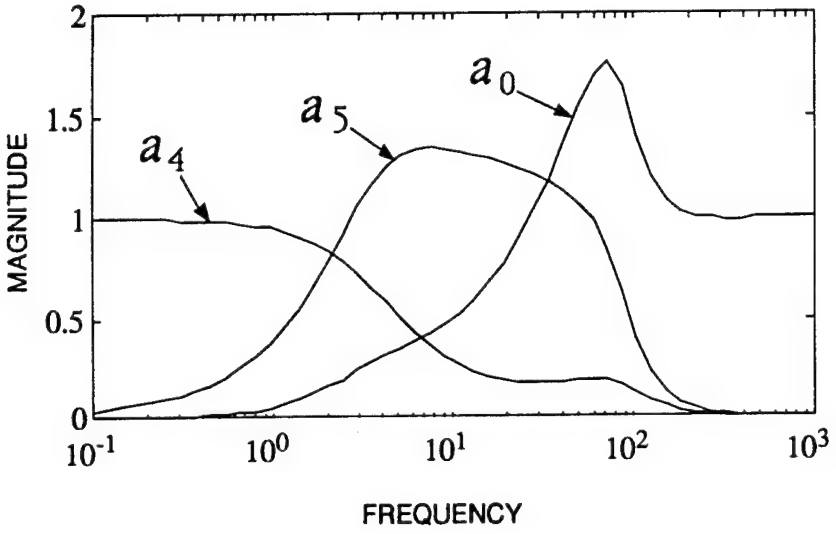


Figure 4.11 Pitch autopilot variation polynomial coefficients versus frequency.

Figure 4.12 shows a surface plot of the magnitude of Eq. (4.12) varying the parameters ϕ_i grouped as $\phi_1 + \phi_2$ and $\phi_1 + \phi_3$. This figure was created for the variation polynomial coefficients evaluated at $\omega = 32.93$ rad/s. This figure shows that the magnitude of the variation polynomial is not a convex function of the ϕ_i (since ϕ_i repeats itself every 2π radians). If a conjugate gradient algorithm is used to find the minimum magnitude of $a(\delta)$, the minimum computed by the algorithm will depend upon where the optimization is started. As Figure 4.12 shows there are an infinite number of minima. Fortunately, we are interested in the *smallest* set of destabilizing parameters, which creates a problem with a unique answer.

Figure 4.13 is a Nyquist plot of the loop transfer function at the plant input. The classical phase margin is 46.32° at a loop gain crossover frequency of 32.93 rad/sec. If a parameter space hypercube is created for the ϕ_i (i=1,2,3), there will be $2^3 = 8$ vertices of this cube. A vertex of this cube maps into the point F_i in Figure 4.13.

In the deGaston-Safonov real stability margin calculation, a straight line in the real parameter space maps into a straight line in the Nyquist plane. Thus, a convex hull formed from the mapped vertices contains the entire image of the hypercube. This is not true for complex uncertainties. As Figure 4.13 shows, a straight line in the ϕ-parameter space maps into an arc in the Nyquist plane. The convex hull formed from the mapped vertices will not contain the mapping of the edges of the hypercube, let alone the entire image of the hypercube. This precludes the use of the same analysis software for both types of uncertainties.

Our optimization results, which minimize the magnitude of Eq. (4.12), are shown in Figure 4.14 and indicate that no combination of the ϕ_i's will destabilize the system at low and high frequencies. Only in a small interval near the loop gain crossover frequency can this system be destabilized. This is shown in Figure 4.14. Figure 4.14 is a plot of the minimum variation polynomial magnitude as a function of frequency.

A multivariable phase margin can be defined as follows:

$$\phi_{PM} = \min_{\phi_{max}} \left\{ \phi_i \in [-\pi,\ \pi],\ -\phi_{max} \le \phi_i \le \phi_{max} /\ det[I - \Delta M] \ne 0\ \forall \omega \right\} \qquad (4.13)$$

This is equivalent to defining a hypercube in the ϕ_i parameter space, expanding this cube, and guaranteeing that all combinations of the ϕ_i interior to this cube result in a stable system. Figure 4.15 shows the destabilizing phase variations plotted as a solid curve in the 3-dimensional ϕ-parameter space, along with the largest cube tangent to the curve. Using Eq (4.13), we have ϕ_{PM}

= 23°. Note that using Eq. (4.13) does not result in the same "smallest destabilizing uncertainty" as using the 2-norm of the parameter uncertainties in a vector.

If the missile autopilot loop gain transfer function $L(s)$ is formed at the actuator input, the transfer function is

$$\begin{aligned} L(s) &= e^{-j\phi_1}\left(e^{-j\phi_3}K_1(s)K_2(s)\frac{A_z}{\delta_c}(s) + e^{-j\phi_2}K_2(s)\frac{q}{\delta_c}(s)\right) \\ &= e^{-j(\phi_1+\phi_3)}K_1(s)K_2(s)\frac{A_z}{\delta_c}(s) + e^{-j(\phi_1+\phi_2)}K_2(s)\frac{q}{\delta_c}(s) \end{aligned} \tag{4.14}$$

Note that the only nonzero coefficients of $a(\delta)$ correspond to the coefficients of $a(\delta)$ that multiply $e^{-j(\phi_1+\phi_2)}$ and $e^{-j(\phi_1+\phi_3)}$. It turns out that the smallest set of destabilizing ϕ's, defined by the point tangent to the cube in Figure 4.15, predict the classical phase margin shown in Figure 4.13. That is $min(\phi_1+\phi_2,\ \phi_1+\phi_3)$ = 46.32°, indicating agreement with the variation polynomial prediction of stability.

To further demonstrate our approach, the variation polynomial approach is presented for a roll-yaw autopilot analysis problem. Like the pitch autopilot analysis problem, the problem is to place a complex exponential $e^{-j\phi}$ in each input and output channel, and determine the smallest set of ϕ's that can destabilize the missile. Figure 4.16 shows a signal flow graph with the phase uncertainties added at the inputs and outputs of this missile system. In this example, the smallest destabilizing uncertainties are found by stacking the ϕ's in a vector, and using the 2-norm of that vector as a measure of size.

Figure 4.17 shows the singular values versus frequency of the loop transfer function matrix without the phase uncertainties. The loop gain crossover frequency, defined to be the largest frequency where the maximum singular value of the loop transfer function matrix is unity (0 dB), is near 18 rad/s. Our results will show that it is in this frequency region, the region near the loop gain crossover frequency, that the parameter uncertainties can destabilize this system.

The variation polynomial for this system is

$$a(\delta) = a^T\delta$$

Where $a^T = [a_0\ \ a_6\ \ a_7\ \ a_{10}\ \ a_{11}\ \ a_{15}]$ and

$$\delta^T = \left[1\ \ e^{-j(\phi_2+\phi_4)}\ \ e^{-j(\phi_2+\phi_3)}\ \ e^{-j(\phi_1+\phi_4)}\ \ e^{-j(\phi_1+\phi_3)}\ \ e^{-j(\phi_1+\phi_2+\phi_3+\phi_4)}\right] \tag{4.14}$$

Only 5 of the $2^4 = 16$ polynomial coefficients are non-zero. The remaining 10 coefficients of a are zero for all frequencies and all angles-of-attack.

Figure 4.18 shows the absolute value of the non-zero coefficients versus frequency. Figure 4.19 presents the minimum value of the objective function $F = \left(a^T \delta\right)(\bullet)^*$ as a function of frequency. Note that only in a small region near the loop gain crossover frequency does the objective function actually become zero. At the other frequencies no combination of the phase parameters destabilize the system (change the number of encirclements of the $det[I - \Delta M]$)

Figure 4.20 shows the 2-norm of the ϕ's (2-norm of $[\phi_1 \quad \phi_2 \quad \phi_3 \quad \phi_4]$) returned from our algorithm at the same objective function values as shown in Figure 4.19. Only in the frequency range between 4 and 20 rad/s can the system be destabilized by these phase variations.

A simple mixed (real and complex) uncertainty model has also been used to test ROBUSTC. The model contains one real parameter uncertainty (δk) and one complex parameter uncertainty $\left(e^{-j\phi}\right)$, and is shown in Figure 4.21. ROBUSTC correctly computed the gain and phase margins for this SISO control system model.

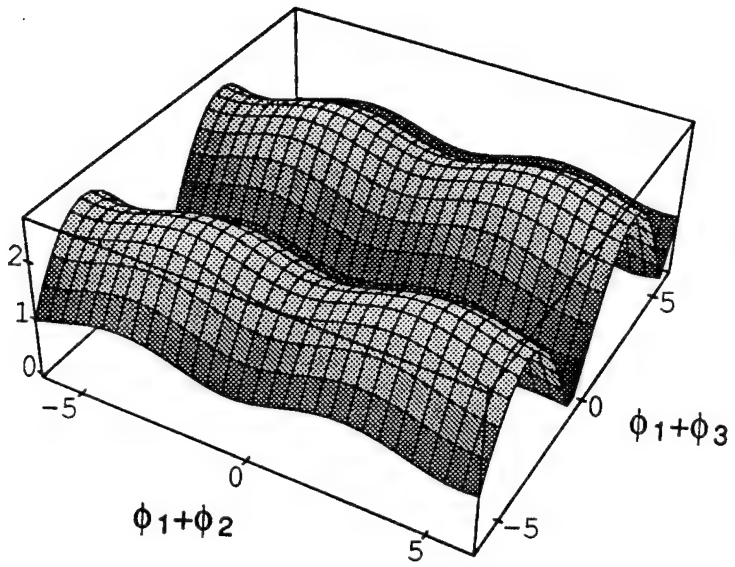


Figure 4.12 Variation polynomial magntiude at ω = 32.93 rad/s.

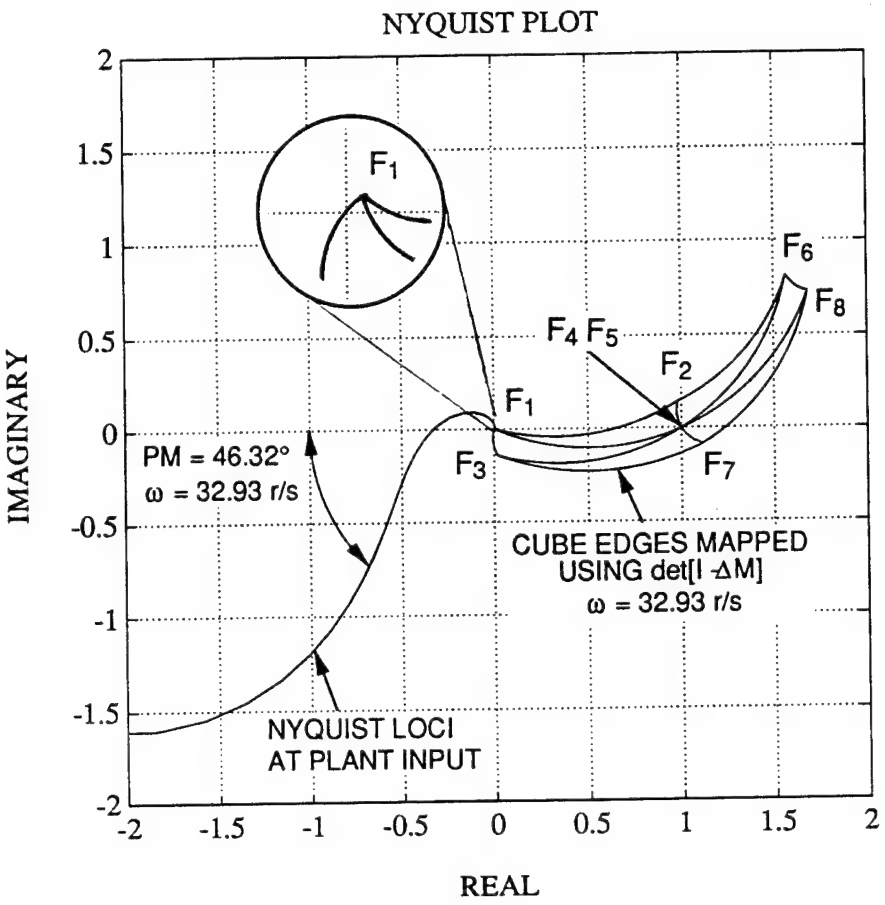


Figure 4.13 Acceleration autopilot Nyquist analysis.

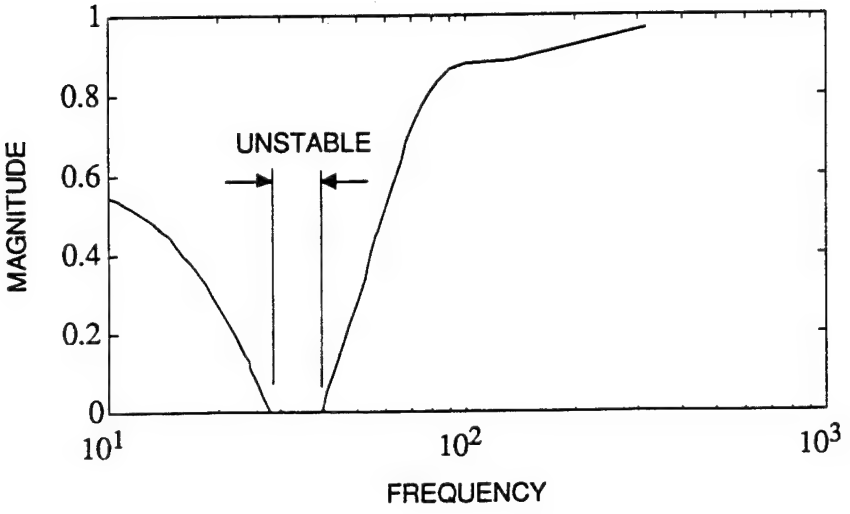


Figure 4.14 Pitch autopilot variation polynomial magntiude.

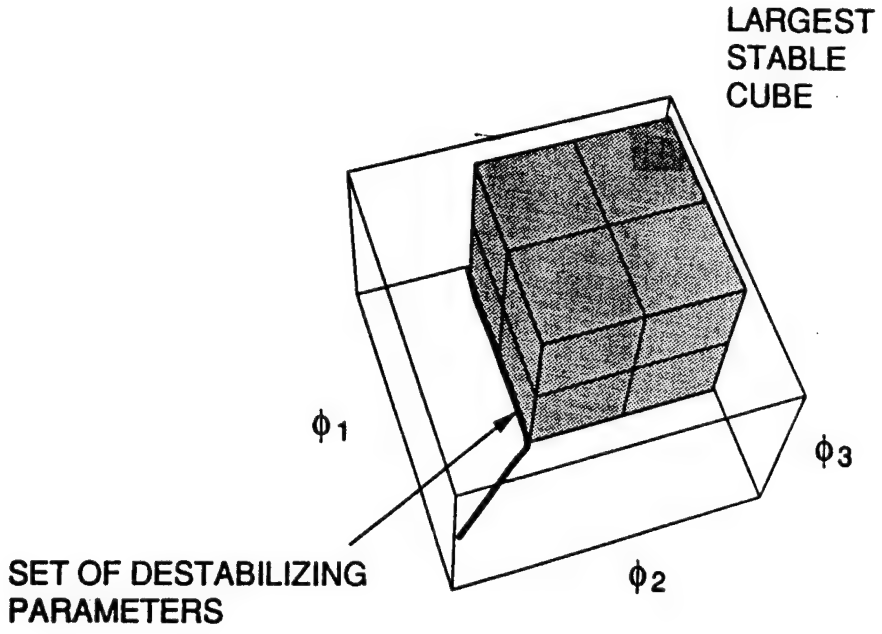


Figure 4.15 Largest stable parameter hypercube.

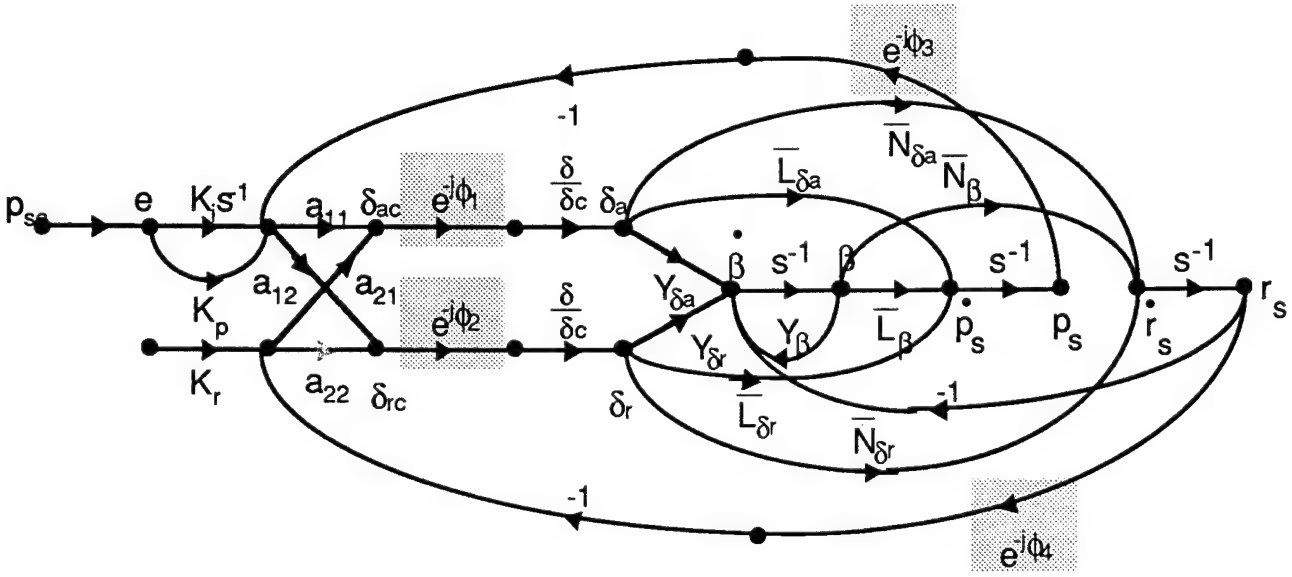


Figure 4.16 Roll-Yaw Signal Flow Graph Analysis Model

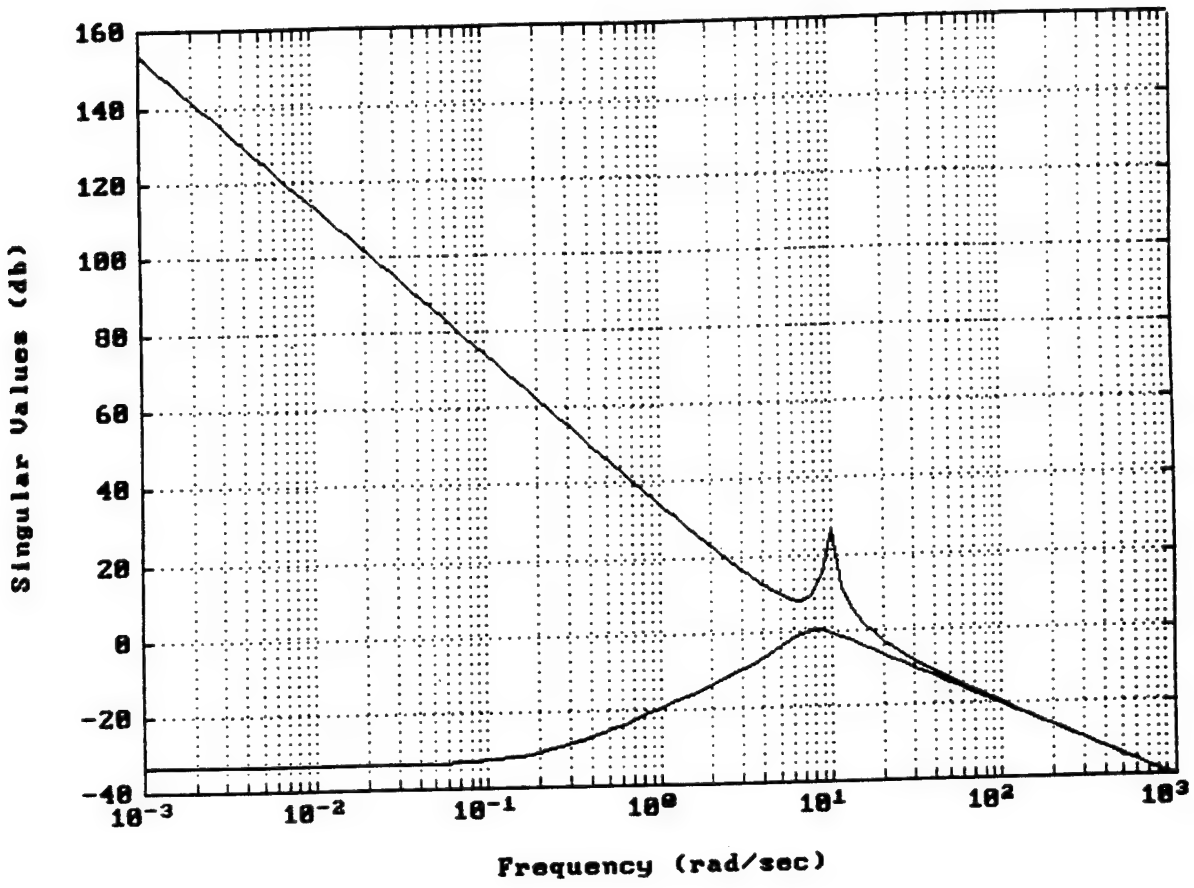


Figure 4.17 Roll-yaw loop gain singular value frequency response.

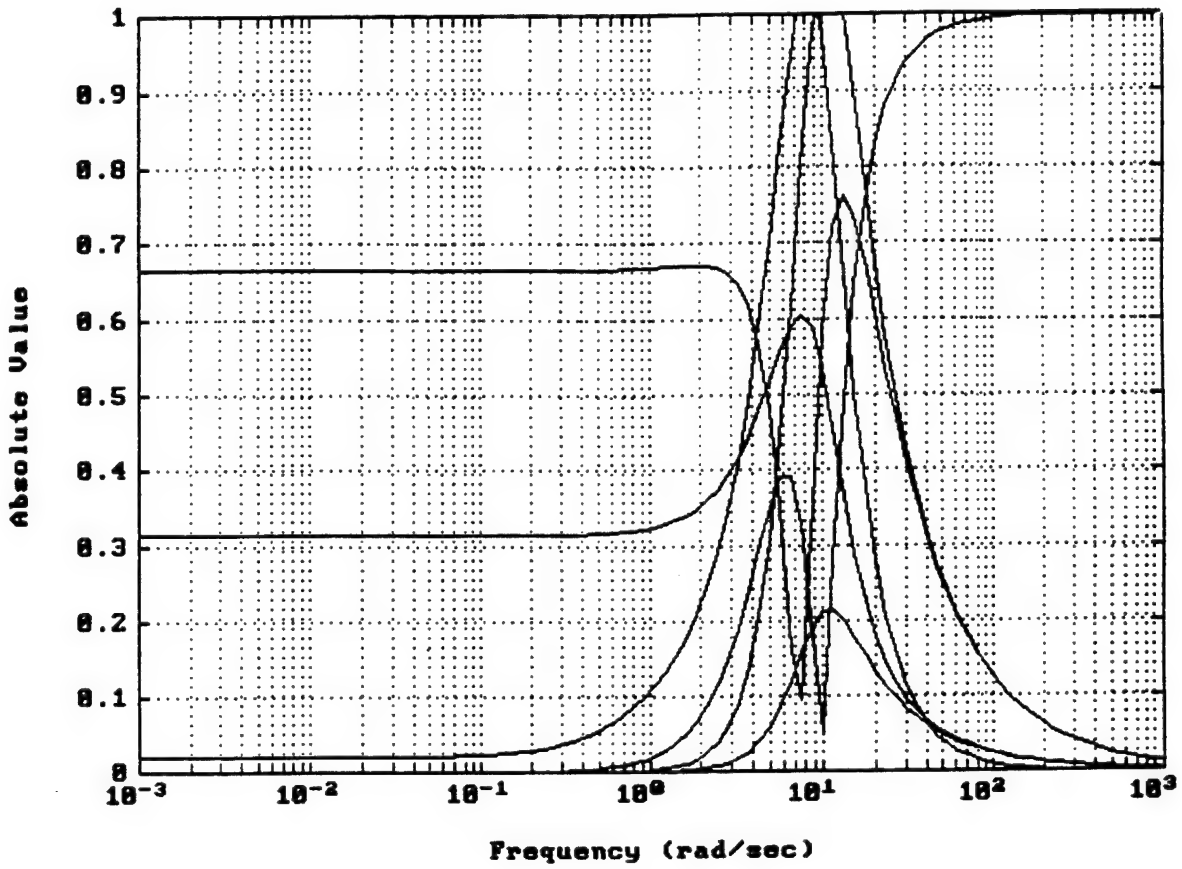


Figure 4.18 Roll-yaw variation polynomial coefficient magnitudes.

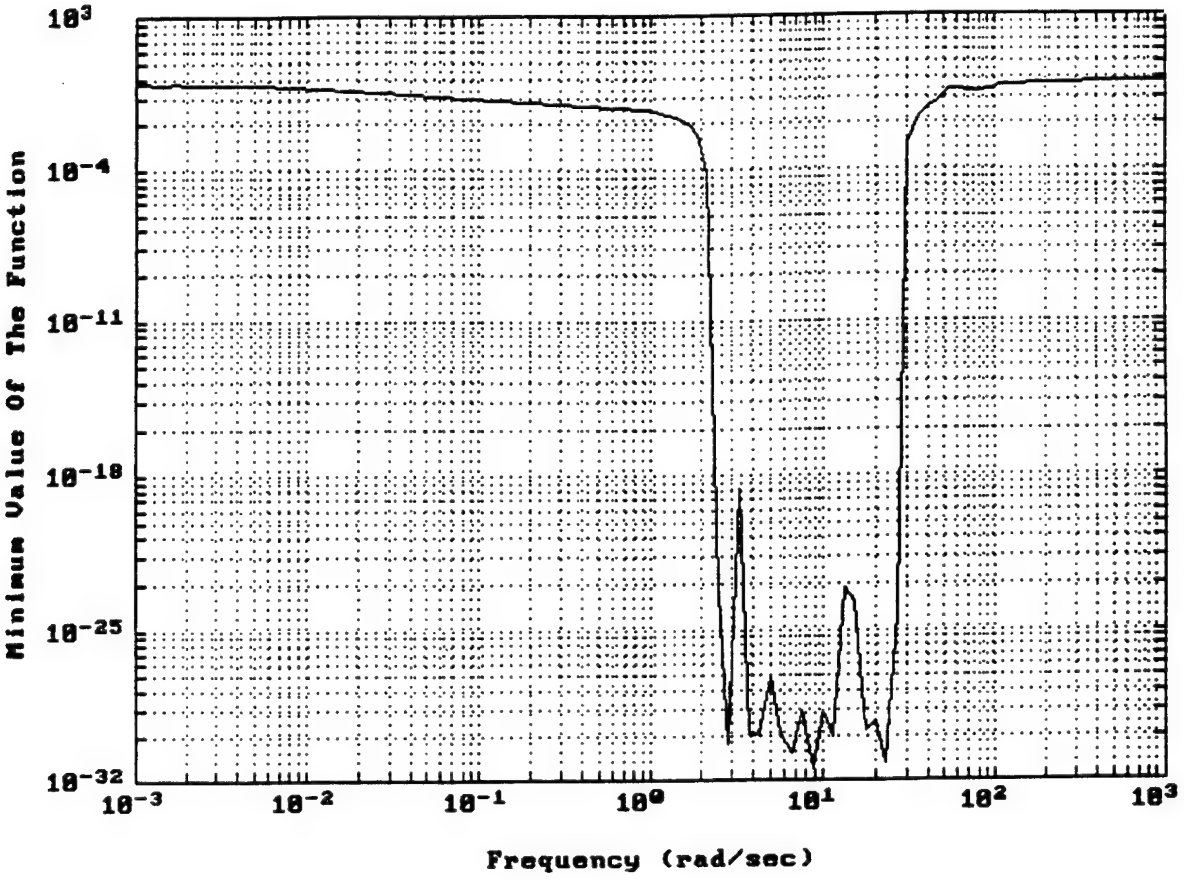


Figure 4.19 Roll-yaw variation polynomial magnitude.

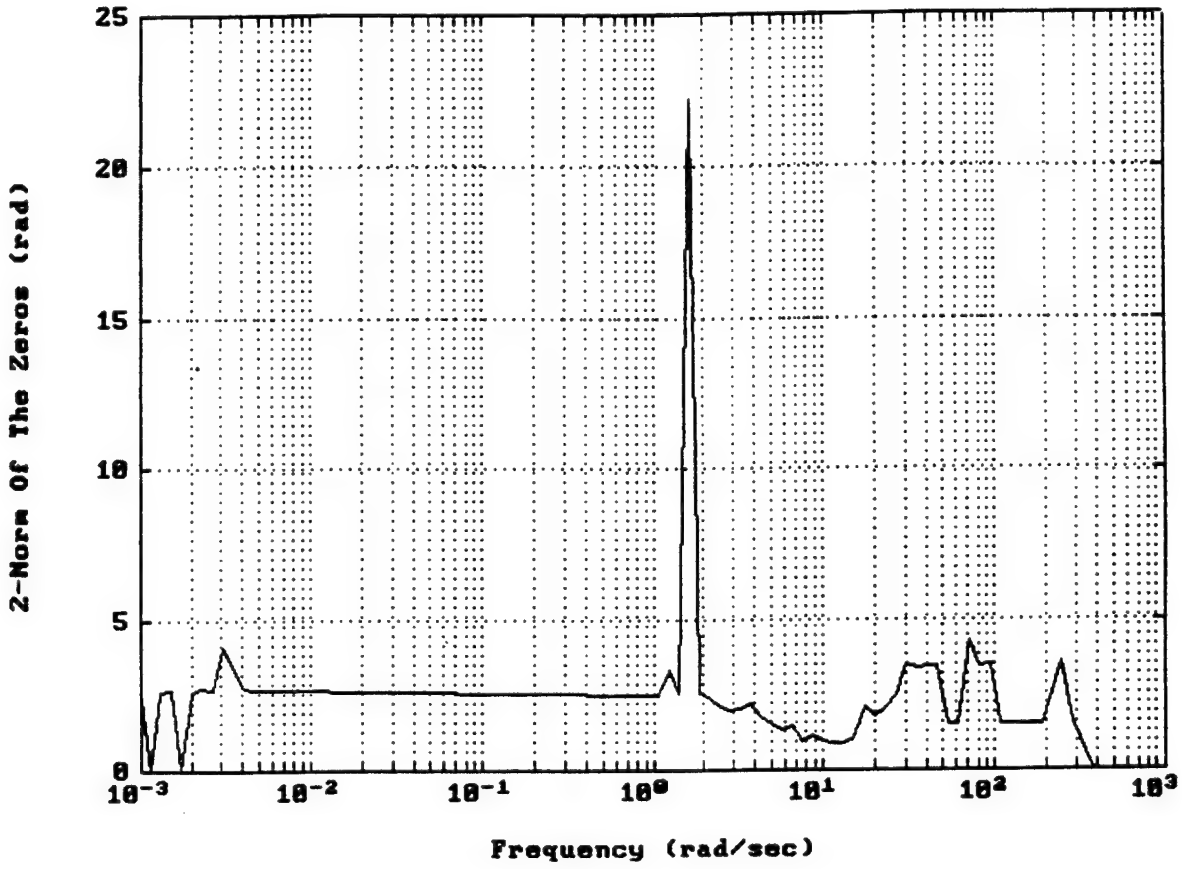
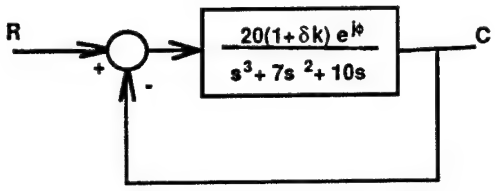
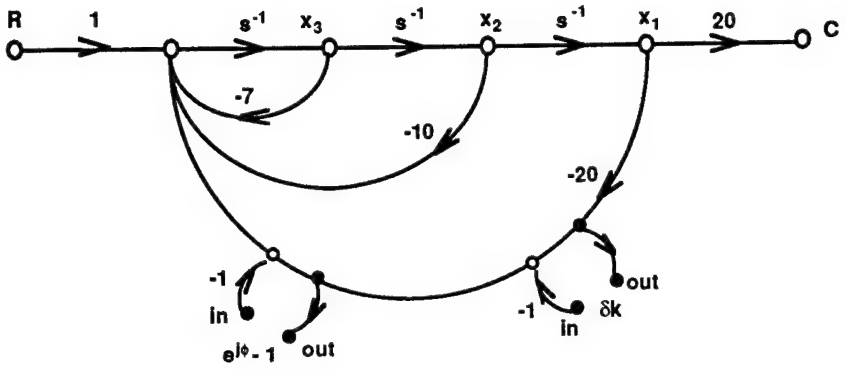


Figure 4.20 2-Norm of the destabilizing ϕ parameter vector.

Mixed Real-Complex Parameter Problem (Gain/Phase Margin Example)





Characteristic Equation

$s^3+7s^2+10s+20(1+\delta k)e^{j\phi}=0$

Analytic Results

Phase margin: 35.79 deg @1.5224 rad/sec

Gain margin: 10.88 dB @ 3.1622776 rad/sec

ROBUSTC Results

@ ω = 3.1622776 rad/sec:

$\delta k = 2.5000$

$e^{j\phi} = -1.7e^{-8}$

@ ω = 1.5224 rad/sec:

$\delta k = 9.6e^{-7}$

$e^{j\phi} = .6246$ rad

Figure 4.21 Gain and phase margin example.

Simulated Annealing Trade Study

In our application of simulated annealing, the annealing algorithm is used to start a conjugate gradient algorithm that minimizes the variation polynomial. The objective function is not a convex function, so local minima can result, depending upon where the algorithm is started. The annealing algorithm, by jumping all over the parameter space for the uncertainties, and starting the conjugate gradient algorithm at each jump, finds the smallest set of destabilizing parameter uncertainties.

Simulated annealing finds the global minimum by jumping around in the parameter space (due to the high temperatures) and evaluating the cost at each jump. The smallest cost and parameters associated with that cost are stored for retrieval. The algorithm stops when the process has sufficiently cooled, as specified by the final temperature.

The choice of the final temperature effects the final outcome of the analysis. If the final temperature is specified at too high a level, the global minimum may not be found. Figure 4.22 illustrates trade study results varying the simulated annealing final temperature. Shown in the figure is a plot of final temperature versus CPU time, analyzing the pitch autopilot analysis problem given in Eq. (4.12). The shaded region illustrates the numerical value (for this problem) where the algorithm generated identical final values for the minimum. If a final temperature was

set higher than this value the smallest set of parameters (using the 2-norm) was not found by the algorithm. Thus, to us simulated annealing effectively, low final temperatures must be used.

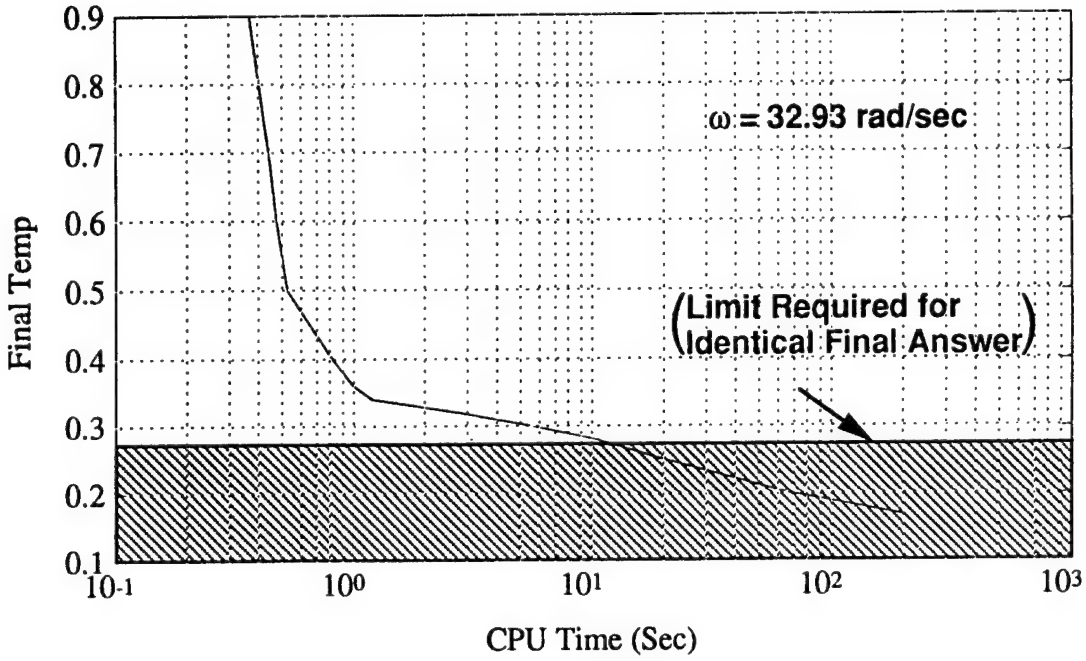


Figure 4.22 Simulated annealing final temperature versus CPU time.

4.4 Conclusions

Our results show that the magnitude of the coefficients of the variation polynomial vary rapidly within a certain range near the loop gain crossover frequency. From the optimization results it is clear that the objective function takes on it's minimum value, which can be considered as zero, in this same frequency range indicating that the system is destabilized by the complex parameter variations. The simulated annealing/conjugate gradient algorithm gives the exact parameter values that destabilize the system.

These results show that it is possible to develop parameter space based analysis tests for dynamic uncertainties, including the class of problems in which the parameters do not appear linearly-fractionally in the analysis model. Thus, the variation polynomial approach gives an exact estimate of complex parameter variations for the system to stable.

Although this algorithm suffers from an exponential explosion in calculations (2^n at least once) we have found that the computation times are very reasonable for problem of interest in aircraft and missile flight control.

4.5 Chapter 4 References

[1]. R. E. Eberhardt and K. A. Wise, "Automated Gain Schedules for Missile Autopilots Using Robustness Theory," Proc. of the 1st IEEE Conference on Control Applications, Dayton, OH, Sept. 1992, pp. 243-250.

[2]. R. deGaston and M. Safonov, "Exact Calculation of the Multiloop Stability Margin", *IEEE Transactions on Automatic Control*, vol. 33, No. 2, pp. 156-171, Feb 1988.

[3]. A. Sideris, "Elimination of Frequency Search from Robustness Tests", *IEEE Transactions on Automatic Control*, vol. 37, No. 10, pp. 1635-1640, Oct. 1992.

[4]. Doyle, J. C., "Robustness of Multiloop Linear Feedback Systems", Proc. of the IEEE CDC, p12-18, 1978.

[5]. Doyle, J. C., "Structured Uncertainty in Control System Design", Proc. of the IEEE CDC, p260-265, 1985.

[6]. Doyle, J. C., J. E. Wall and G. Stein, "Performance and Robustness Analysis for Structured Uncertainty", Proc. of the 22nd CDC, pp. 629-636, 1982.

[7]. Young, P., M. Newlin, and J. C. Doyle, "Practical Computation of the Mixed μ Problem," Proc. of the ACC, Chicago, IL, May 1992.

[8]. Klein, R. E., "Teaching Linear Systems Theory Using Cramer's Rule," IEEE Trans. on Education, Vol. 33, No. 3, August, 1990, pp. 258-267.

[9]. Wise, K. A., B. C. Mears, C. Tang, and A. Godhwani, "A Convex Hull Program Evaluating Control System Robustness To Real Parameter Variations," *Proc. of the AIAA Guidance, Navigation, and Control Conference*, Portland, OR, August, 1990, pp. 223-231.

[10]. Wise, K. A., "Missile Autopilot Robustness Using the Real Multiloop Stability Margin," *Journal of Guidance, Control, and Dynamics*, Vol. 16, No. 2, March-April, 1993, pp. 354-362.

[11]. Kirk, D. E., *Optimal Control Theory, An Introduction*, Prentice-Hall, New Jersey, 1970.

[12]. Wismer, D. A., and R. Chattergy, *Introduction to Non-linear Optimization, A problem solving Approach*, North Holland, New York, 1979.

[13]. Fox, R. L., *Optimization Methods for Engineering Design*, Addison-Wesley, CA, 1971.

[14]. Hestenes, M., *Conjugate Direction Methods in Optimization* , Springer-Verlag, New York, 1980.

[15]. Arts, E. and J. Korst, Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing, John Wiley & Sons, New York, 1989.

[16]. Wise, K. A., and S. Kundur, "Missile Autopilot Robustness to Real and Complex Uncertainties Using A Parameter Space Robustness Test," Proc of the AIAA Guidance, Navigation, and Control Conference, Monterey, CA, August, 1993, pp. 336-346.

[17]. Wise, K. A., "A Parameter Space Robustness Test for Real and Complex Uncertainties," Proc. of the 32nd IEEE Conference on Decision and Control Theory, San Antonio, TX, Dec. 1993, pp. 2486-2492.

Appendix A

Nonlinear H_∞ Software Documentation

The software that generates the nonlinear H_∞ control is applicable to nonlinear systems of the form

$$\begin{aligned} \dot{x} &= Ax + \Delta f(x) + G_1 u + G_2 w \\ z &= Cx + D_1 u + D_2 w \end{aligned} \tag{A.1}$$

where $\Delta f(x)$ is $O\left(x^2\right)$. For the nonlinear system described in Eq. (A.1), the HJI equation for optimal state feedback nonlinear H_∞ control is

$$x^T C^T C x - x^T C^T S R^{-1} S^T C x + V_x^*\left(Ax + \Delta f(x) - B R^{-1} S^T C x\right) - \tfrac{1}{4} V_x^* B R^{-1} B^T V_x^{*T} = 0 \tag{A.2}$$

where

$$B = \begin{bmatrix} G_1 & G_2 \end{bmatrix}$$

$$S = \begin{bmatrix} D_1 & D_2 \end{bmatrix}$$

$$R = \begin{bmatrix} D_1^T D_1 & D_1^T D_2 \\ D_2^T D_1 & D_2^T D_2 - \gamma^2 I \end{bmatrix}$$

and γ is the attenuation level given by a linear H_∞ gain scheduled control design.

Let $V(x)$ represent a Lyapunov function of the form

$$V(x) = x^T X x + \Delta V(x) \tag{A.3}$$

with $\Delta V(x) = O\left(x^3\right)$. Using this definition for $V(x)$, substitute the gradient $V_x(x)$ into the HJI equation in Eq. (A.2). The result is

$$x^T\left(X\tilde{A} + \tilde{A}^T X + \tilde{Q} + X\tilde{R}X\right)x + 2x^T X \Delta f + \Delta V_x\left[\tilde{F}x + \Delta f\right] + \tfrac{1}{4}\Delta V_x \tilde{R} \Delta V_x^T = 0$$

where

$$\tilde{A} = A - BR^{-1}S^T C$$
$$\tilde{Q} = C^T\left(I - SR^{-1}S^T\right)C$$
$$\tilde{R} = -BR^{-1}B^T$$
$$\tilde{F} = \tilde{A} + \tilde{R}X$$

Let X be the solution to the (linear) Algebraic Riccati Equation (ARE):

$$X\tilde{A} + \tilde{A}^T X + \tilde{Q} + X\tilde{R}X = 0. \tag{A.4}$$

Then

$$x^T\left(\underbrace{X\tilde{A} + \tilde{A}^T X + \tilde{Q} + X\tilde{R}X}_{=0}\right)x + 2x^T X\Delta f + \Delta V_x\left[\tilde{F}x + \Delta f\right] + \tfrac{1}{4}\Delta V_x\tilde{R}\Delta V_x^T = 0 \tag{A.5}$$

which simplifies to

$$2x^T X\Delta f + \Delta V_x\left[\tilde{F}x + \Delta f\right] + \tfrac{1}{4}\Delta V_x\tilde{R}\Delta V_x^T = 0. \tag{A.6}$$

Solving this PDE for ΔV, and combining with the linear part in Eq. (A.4) yields the optimal state feedback nonlinear H_∞ control given by

$$u(x) = [1 \quad 0]\left(-R^{-1}\right)\left\{\tfrac{1}{2}B^T\left(2Xx + \Delta V_x^T\right) + S^T Cx\right\} \tag{A.7}$$

The characteristic equations for the solution of the PDE in Eq. (A.6) are

$$\begin{aligned}\frac{dz}{dt} &= \tilde{F}z + \tfrac{1}{2}\tilde{R}p + \Delta f(z)\\ \frac{dp}{dt} &= -\tilde{F}^T z - 2\Delta f_z^T Xz - \Delta f_z^T p - 2X\Delta f(z)\end{aligned} \tag{A.8}$$

Solutions to these characteristic equations can be represented in integral form as

$$\begin{aligned}z(t) = e^{\tilde{F}t}\Bigg\{x - \hat{\rho}\int_t^\infty e^{\tilde{F}^T\tau}q(z,p)d\tau + \int_0^t\left(e^{-\tilde{F}\tau}\left(\Delta f(z) + \hat{\rho}q(z,p)\right) - \hat{\rho}e^{\tilde{F}\tau}q(z,p)\right)d\tau\Bigg\}\\ + \hat{\rho}e^{-\tilde{F}^T t}\int_t^\infty e^{\tilde{F}^T\tau}q(z,p)d\tau\end{aligned} \tag{A.9}$$

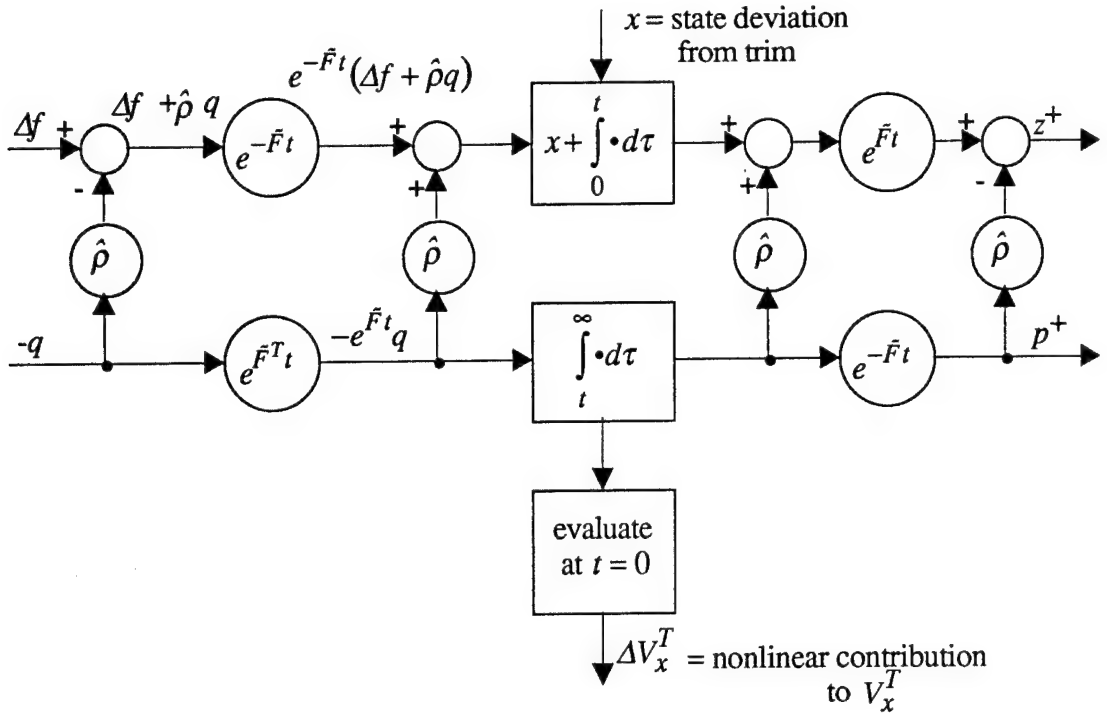


Figure A.1 Computational flow computing ΔV_x^T

$$p(t) = -e^{-\tilde{F}^T t}\int_t^\infty e^{\tilde{F}^T \tau} q(z,p)\,d\tau$$

where

$$q(z,p) = -\Delta f_z^T(z)(2Xz + p) - 2X\Delta f(z) \qquad \text{(A.10)}$$

and $\hat{\rho}$ satisfies the Lyapunov equation $\tilde{F}\hat{\rho} + \hat{\rho}\tilde{F}^T = \frac{1}{2}\tilde{R}$. It was shown earlier that iterations defined by evaluation of the right hand side of the integral equations defines a local contraction mapping.

ΔV_x^T is then given by

$$\Delta V_x^T = p(0) \qquad \text{(A.11)}$$

and the feedback control is given by Eq. (A.7).

The computational flow corresponding to the iterations can be diagrammed as shown in Figure A.1. This structure is independent of the specific form of the nonlinearity Δf. The iteration is initialized at $z = x$, $p = 0$, at the start of operation, and can run continuously thereafter. The changing value of x then drives the system to provide changes in ΔV_x^T.

Alternatively, the computation can be initialized to zero at each time and a preset number of iterations can be generated. The nonlinear calculation needed in the iterative procedure is considered next.

The nonlinear calculations for Δf and $-q$ are problem dependent. We show the computation of Δf and $-q$ for two examples, a 2nd order system with quadratic nonlinearity, and then our sixth order missile pitch autopilot design problem. These examples illustrate how the software calculates the parts of the integral expressions that are problem dependent.

Second Order System With Quadratic Nonlinearity

In this case we consider

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \Delta f(z) = \begin{bmatrix} c_1 z_1^2 \\ 0 \end{bmatrix} \tag{A.12}$$

Define $e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we can express Δf as

$$\begin{aligned} \Delta f(z) &= c_1 \left(e_1^T z\right)\left(e_1 e_1^T\right) z \\ &= \left(e_1^T z\right) G z \end{aligned} \tag{A.13}$$

where $G = \begin{bmatrix} c_1 & 0 \\ 0 & 0 \end{bmatrix}$. If we let $w = 2Xz + p$, then

$$\begin{aligned} -q &= 2X\Delta f + \Delta f_z^T w \\ \Delta f_z^T &= 2c_1 \left(e_1^T z\right) e_1 e_1^T \end{aligned} \tag{A.14}$$

and

$$\begin{aligned} \Delta f_z^T w &= 2c_1 \left(e_1^T w\right) e_1 e_1^T z \\ &= \left(e_1^T w\right) H z \end{aligned} \tag{A.15}$$

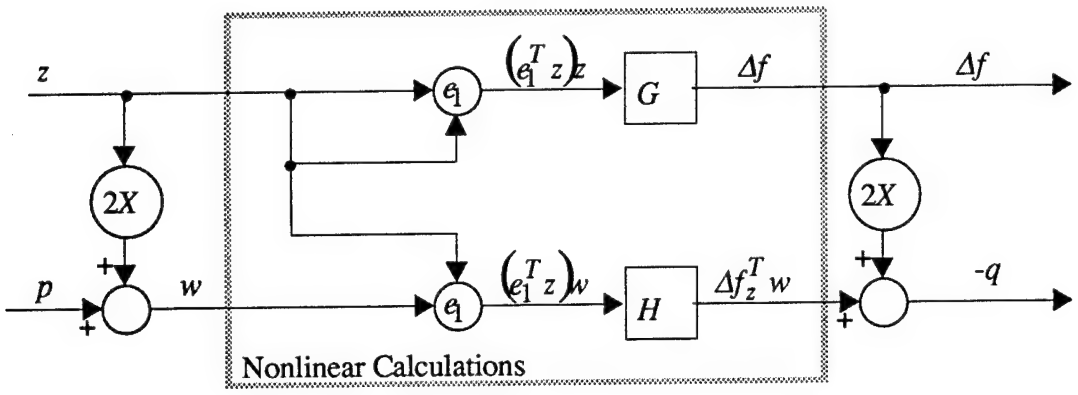


Figure A.2 Calculation of the nonlinearities for the 2nd order problem.

where $H = \begin{bmatrix} 2c_1 & 0 \\ 0 & 0 \end{bmatrix}$. A computational flow for calculating Δf and $-q$ for this example is shown in Figure A.2.

Sixth Order Missile Pitch Autopilot Problem

Consider the missile autopilot problem discussed in Section 2.2. For this problem we have

$$\begin{aligned} z^T &= [z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6] \\ g^T &= [c_1 \quad 0 \quad c_2 \quad 0 \quad 0 \quad 0] \\ h^T &= [c_3 \quad 0 \quad c_4 \quad 0 \quad 0 \quad 0] \\ e_1^T &= [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \end{aligned} \tag{A.16}$$

and that

$$\begin{aligned} \Delta f(z) &= \left(c_1 z_1^2 + c_2 z_1 z_3 + c_3 z_1^3 + c_4 z_1^2 z_3\right) e_1 \\ &= \left(\left(g^T z\right) z_1 + \left(h^T z\right) z_1^2\right) e_1 \\ &= \left(\left(g^T z\right)\left(e_1^T z\right) + \left(h^T z\right)\left(e_1^T z\right)^2\right) e_1 \end{aligned} \tag{A.17}$$

To compute the gradient of Δf with respect to the state vector z, Δf_z, differentiate the above expression. This yields

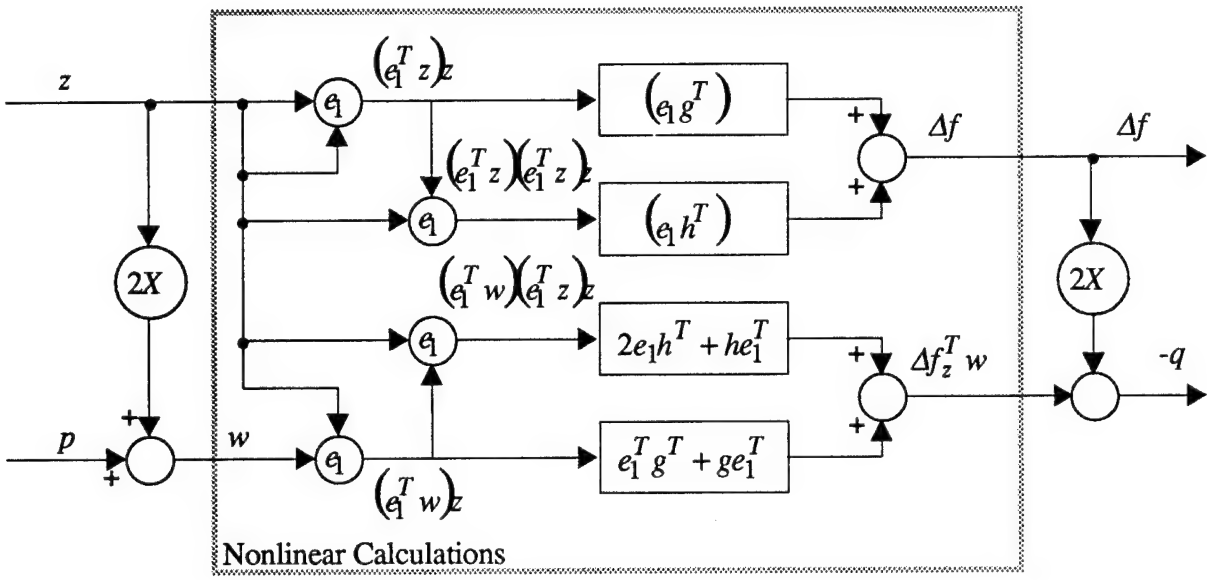


Figure A.3 Calculation of the nonlinearities for the 6th order missile problem.

$$\Delta f_z(z) = e_1\left(\left(g^T z\right)e_1^T + \left(e_1^T z\right)g^T + 2\left(h^T z\right)\left(e_1^T z\right)e_1 + \left(e_1^T z\right)^2 h^T \right)$$
$$\Delta f_z^T(z) = \left(g^T z\right)e_1 e_1^T + \left(e_1^T z\right)g e_1^T + 2\left(h^T z\right)\left(e_1^T z\right)e_1 e_1^T + \left(e_1^T z\right)^2 h^T e_1^T \qquad \text{(A.18)}$$

Multiplying by w results in

$$\begin{aligned}\Delta f_z^T(z) &= \left(g^T z\right)\left(e_1^T w\right)e_1 + \left(e_1^T z\right)\left(e_1^T w\right)g + 2\left(h^T z\right)\left(e_1^T z\right)\left(e_1^T w\right)e_1 + \left(e_1^T z\right)^2\left(e_1^T w\right)h \\ &= \left(e_1^T w\right)\left(e_1 g^T\right)z + \left(e_1^T w\right)\left(g e_1^T\right)z + 2\left(e_1^T z\right)\left(e_1^T w\right)\left(e_1 h^T\right)z + \left(e_1^T z\right)\left(e_1^T w\right)\left(h e_1^T\right)z \\ &= \left(e_1^T g^T + g e_1^T\right)\left(e_1^T w\right)z + \left(2e_1 h^T + h e_1^T\right)\left(e_1^T z\right)\left(e_1^T w\right)z\end{aligned} \qquad \text{(A.19)}$$

An implementation of this calculation is shown in Figure A.3.

Software Implementation

As discussed in Section 2.2, a modal representation of the matrix exponentials is used in the integral equations (for z and p). This representation generates linear combinations of exponentials with vector polynomial coefficients. The FORTRAN implementation of the successive approximation represents these linear combinations of exponentials with vector

polynomial coefficients using a pair of arrays. The coefficients are stored in a 3-dimensional array, with the exponents of the exponentials stored in a 1-dimensional array.

For example, consider the following linear combination of exponentials with vector polynomial coefficients

$$p = \left[\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}2\\1\end{bmatrix} t\right] e^{-t} + \left[\begin{bmatrix}2\\3\end{bmatrix} + \begin{bmatrix}0\\0\end{bmatrix} t\right] e^{-3t} \tag{A.20}$$

This would be represented by the arrays P and $PEXP$ as

$$\begin{array}{ll} P[1,1,1]=1 & P[1,1,2]=2 \\ P[2,1,1]=0 & P[2,1,2]=3 \\ P[1,2,1]=2 & P[1,2,2]=0 \\ P[2,2,1]=1 & P[2,2,2]=0 \end{array} \tag{A.21}$$

$$PEXP[1,]=-1 \quad PEXP[2]=-3$$

The inclusion of the powers of t in the coefficients is required to handle multiple eigenvalues and to handle integrals of expressions (Eq. (A.9)). The range of the indices will grow as the computation proceeds, because of the multiplications due to the nonlinearities and the matrix exponential gains shown in Figure A.1. The ranges of the indices must be monitored during the computation to anticipate excessive array lengths.

Procedures for clustering nearly identical terms and removal of insignificant terms must be implemented so that the length of the arrays does not grow too large. These thinning operations occur immediately after the addition and multiplication operations.

The matrix exponential $e^{\tilde{F}t}$ is represented by a linear combination of exponentials with matrix polynomial coefficients (interpolating polynomials). This is a fixed length linear combination. It also has to be updated as $\tilde{F}$ changes. Computation arrays for $e^{-\tilde{F}t}$, $e^{-\tilde{F}^T t}$,and $e^{\tilde{F}^T t}$ are not needed since their coefficients and exponents can be inferred from the $e^{\tilde{F}t}$ arrays. Calculation of the matrix exponential as arrays is needed for our solution approach, and is shown in Figure A.1. This calculation is accomplished once the eigenvalues of $\tilde{F}$ are provided using the Cayley-Hamilton Theorem.

The real symmetric matrix $\hat{\rho}$ is also needed. This requires the solution of the $n \times n$ Lyapunov equation

$$\tilde{F}\hat{\rho} + \hat{\rho}\tilde{F}^T = \frac{1}{2}\tilde{R} \tag{A.22}$$

where n is the dimension of the state. This solution is accomplished by transforming Eq. (A.22) into a set of $n(n+1)/2$ linear equations in the components of $\hat{\rho}$ and solving by the LU-decomposition.

The solution of the Algebraic Riccati Equation (ARE)

$$X\tilde{A} + \tilde{A}^T X + \tilde{Q} + X\tilde{R}X = 0$$

is also needed for the nonlinear computations (shown in Figure A.3). Our implementation considers the matrices $\tilde{A}, B, \tilde{Q}, \tilde{R}$ as inputs and includes the eigenvalue computations on-line. This approach was used because we have found that the eigenvalues and eigenvectors for H_∞ problems are usually too sensitive to succumb to interpolation.

Figure A.4 summarizes the computations made implementing our successive approximation solution to the integral expressions.

The following 2nd order example is used to further illustrate the computations. Consider the 2nd order example (quadratic nonlinearity in the first state equation) with linearized matrices

$$\tilde{A} = \begin{bmatrix} -4 & 0 \\ 1 & -1 \end{bmatrix},\ B = I_2,\ R = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},\ \tilde{Q} = \begin{bmatrix} 14 & 2 \\ 2 & 2 \end{bmatrix},\ x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

and $c_1 = 0.03$. From the linear H_∞ ARE we have $X = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}$, which gives $\tilde{F} = \begin{bmatrix} -1 & 1 \\ 0 & -2 \end{bmatrix}$, with the matrix $\hat{\rho} = \begin{bmatrix} \frac{-5}{24} & \frac{1}{24} \\ \frac{1}{24} & \frac{3}{24} \end{bmatrix}$. The matrices $G = \begin{bmatrix} 0.03 & 0 \\ 0 & 0 \end{bmatrix}$ and $H = \begin{bmatrix} 0.06 & 0 \\ 0 & 0 \end{bmatrix}$. The matrix exponential $e^{\tilde{F}t}$ is given by

$$e^{\tilde{F}t} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} e^{-t} + \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix} e^{-2t}$$

The software uses a vector exponential string Z to represent $z(t)$ and $\Delta f(t)$ and Q to represent $-p(t)$, $2Xz(t) + p(t)$, etc.

The duplication is advised because in most problems the array length can become very large and one must economize on memory space. The nonlinearity $\Delta f(t)$ is represented by the array DF. Only one temporary array ($TEMP$) is used. For some applications this may not be sufficient so that other temporary arrays must be declared in those cases.

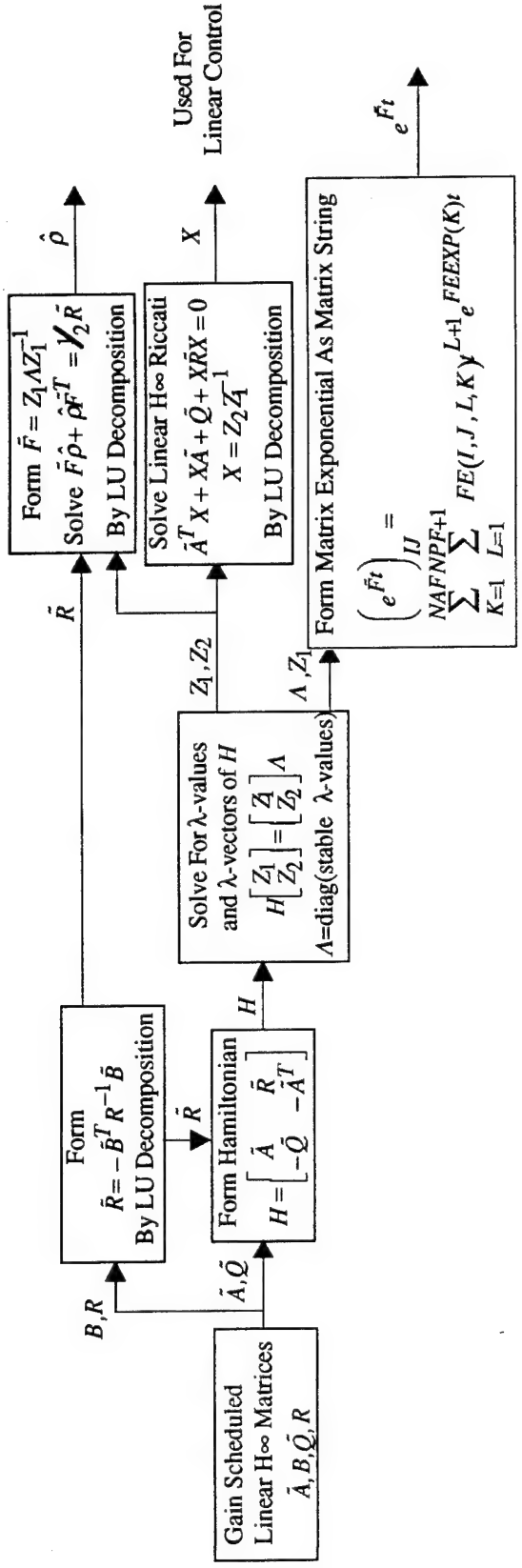


Figure A.4 NLHINF computational flow chart.

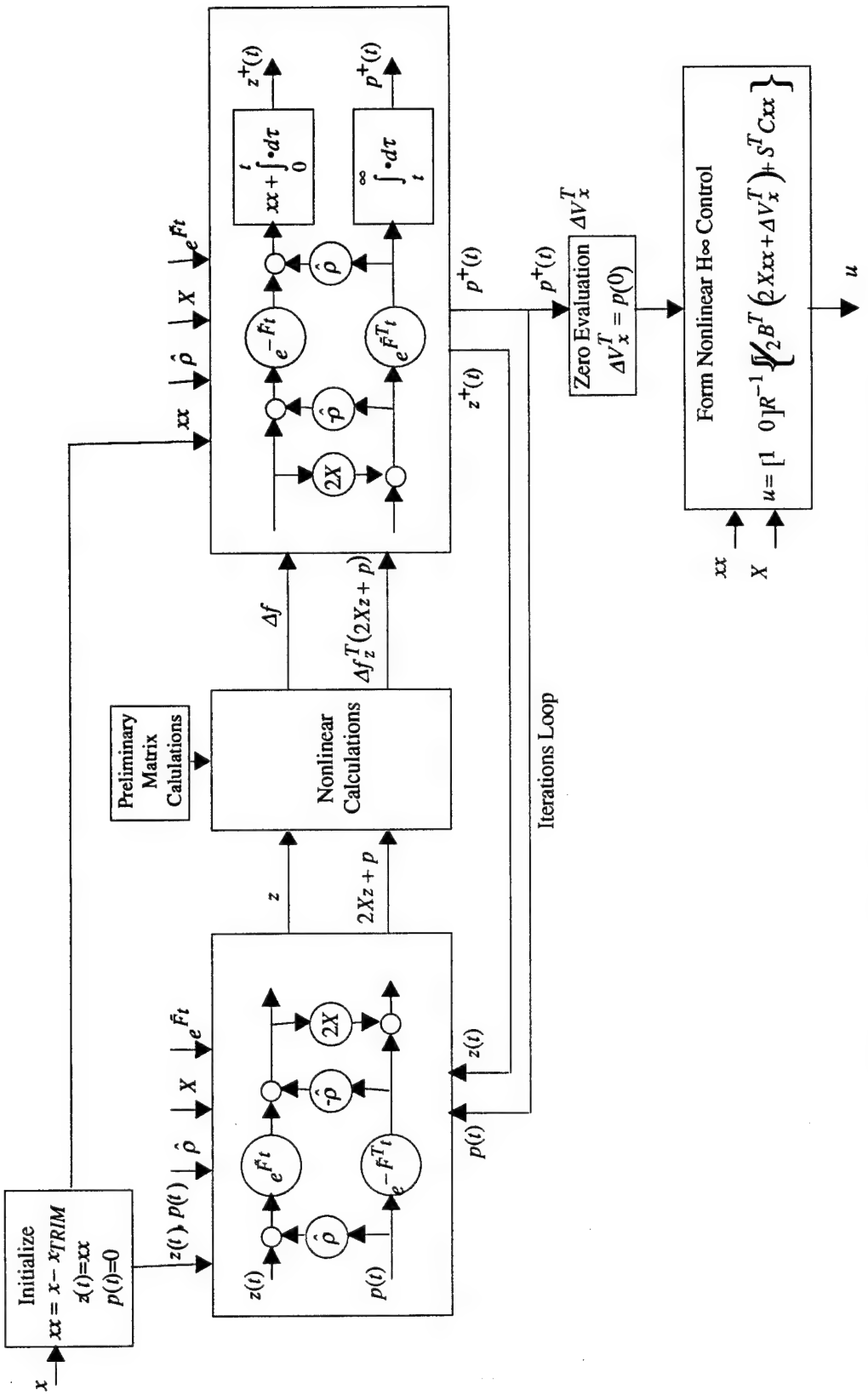


Figure A.4 (Continued) NLHINF computational flow chart.

The software uses various subroutines to combine strings of exponentials or to integrate them. Routines *ADD* and *MULT* are used for the parts of the software which are independent of the nonlinearity.

Subroutine *ADD* multiplies a vector string $A(t)$ by a real constant matrix $C(t)$ time a real scalar coefficient *coef*, and adds it to a string $B(t)$, and returns the result in the string $B(t)$. Figure A.5 illustrates the computational flow for *ADD*.

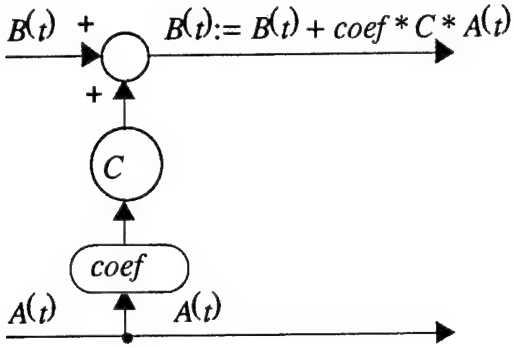


Figure A.5 Computational flow for subroutine *ADD*

The subroutine *MULT* multiplies a vector string $B(t)$ by a matrix string $A(t)$, and returns the result in the string $B(t)$.

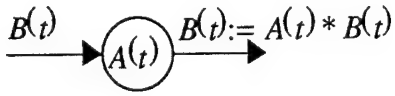


Figure A.6 Computational flow for subroutine *MULT*

The routine gives the options of multiplying by $A(t)$, $A(-t)$, $A^T(t)$, or $A^T(-t)$ so that only the one string $A(t)$ needs to be defined in the software. Figure A.6 illustrates the computational flow for *MULT*.

Subroutine *INT_T_INF* integrates a vector string from t to ∞. These integrals are assumed to exist in our applications because of the stability of $\tilde{F}$, which is the linear H_∞ closed loop matrix. The routine evaluates expressions of the form

$$\int_t^\infty \left(a_1 + a_2\tau + \cdots + a_{m+1}\tau^m\right)e^{\lambda\tau}d\tau = \left(b_1 + b_2 t + \cdots + b_{m+1}t^m\right)e^{\lambda t}$$

with $Re(\lambda) < 0$. Differentiating yields

$$-\left(a_1 + a_2 t + \cdots + a_{m+1}t^m\right)e^{\lambda t} = \left(b_2 + 2b_3 t + \cdots + mb_{m+1}t^{m-1}\right)e^{\lambda t} + \lambda\left(b_1 + b_2 t + \cdots + b_{m+1}t^m\right)e^{\lambda t}$$

Equating like powers of t yields

$$\begin{aligned}
-a_1 &= b_2 + \lambda b_1 \\
-a_2 &= 2b_3 + \lambda b_2 \\
&\vdots \\
-a_m &= mb_{m+1} + \lambda b_m \\
-a_{m+1} &= \lambda b_{m+1}
\end{aligned}$$

and results in the following algorithm

$$\begin{aligned}
a_{m+1} &:= -a_{m+1}/\lambda \\
a_m &:= -(a_m + ma_{m+1})/\lambda \\
&\vdots \\
a_2 &:= -(a_2 + 2a_3)/\lambda \\
a_1 &:= -(a_1 + a_2)/\lambda
\end{aligned}$$

which returns the result in the same array as the input.

Subroutine *INT_0_T* integrates a vector string from 0 to t, then adds the constant vector x to the result and returns the result in the same array as the input. The algorithm is derived in a manner similar to that of *INT_T_INF*.

If $\lambda \neq 0$,

$$
\begin{aligned}
a_{m+1} &:= a_{m+1} / \lambda \\
a_m &:= (a_m + m a_{m+1}) / \lambda \\
&\vdots \\
a_2 &:= (a_2 + 2a_3) / \lambda \\
a_1 &:= (a_1 + a_2) / \lambda
\end{aligned}
$$

If $\lambda = 0$,

$$
\begin{aligned}
a_{m+2} &:= a_{m+1} / m+1 \\
a_{m+1} &:= a_m / m \\
&\vdots \\
a_3 &:= a_2 / 2 \\
a_2 &:= a_1 \\
a_1 &:= 0
\end{aligned}
$$

The vector x is then added to this result.

For the calculation of the nonlinearities Δf and $\Delta f_z^T (2Xz + p)$, several utility routines (*MMULT*, *MULT*3, *ADD*2) are provided which should be sufficient for incorporating most nonlinearities.

Subroutine *MMULT* multiplies a vector string $A(t)$ times a constant real matrix C and returns the result in $A(t)$. Figure A.7 illustrates the computational flow for *MMULT*.

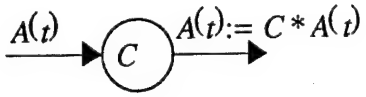


Figure A.7 Computational flow for subroutine *MMULT*

Subroutine *MULT*3 evaluates a matrix string expression of the form

$$C(t) = \left(D^T A(t)\right) B(t)$$

where $A(t)$ and $B(t)$ are matrix strings and D is a real vector.

Subroutine *ADD2* multiplies vector strings $A(t)$ and $B(t)$ by $coef_A C_A$ and $coef_B C_B$, respectively, where $coef_A$ and $coef_B$ are real scalars and C_A and C_B are real matrices. The results are then added and returned in $B(t)$. Figure A.8 illustrates the computational flow for *ADD2*.

The following calculations continue the presentation of the 2nd order example using the subroutines described above. This closely follows the programming in the FORTRAN program *NLHINF* diagrammed in Figure A.4.

Starting with the following matrices:

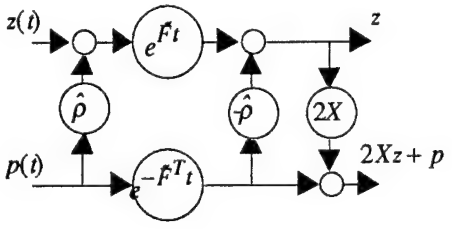
$$\tilde{A} = \begin{bmatrix} -4 & 0 \\ 1 & -1 \end{bmatrix},\ B = I_2,\ R = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},\ \tilde{Q} = \begin{bmatrix} 14 & 2 \\ 2 & 2 \end{bmatrix},\ x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$c_1 = 0.03,\ G = \begin{bmatrix} 0.03 & 0 \\ 0 & 0 \end{bmatrix},\ H = \begin{bmatrix} 0.06 & 0 \\ 0 & 0 \end{bmatrix},$$

$$X = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix},\ \tilde{F} = \begin{bmatrix} -1 & 1 \\ 0 & -2 \end{bmatrix},\ \hat{\rho} = \begin{bmatrix} \frac{-5}{24} & \frac{1}{24} \\ \frac{1}{24} & \frac{3}{24} \end{bmatrix}$$

$$e^{\tilde{F}t} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} e^{-t} + \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix} e^{-2t}$$

The numerical value of the state vector is $xx = [1 \quad 2]^T$. This is used to initialize $z(t)$ in the following flow chart. The initial costate is $p(t) = [0 \quad 0]^T$.



The output from these computations are:

$$z = \begin{bmatrix} 3 \\ 0 \end{bmatrix} e^{-t} + \begin{bmatrix} -2 \\ 2 \end{bmatrix} e^{-2t}$$

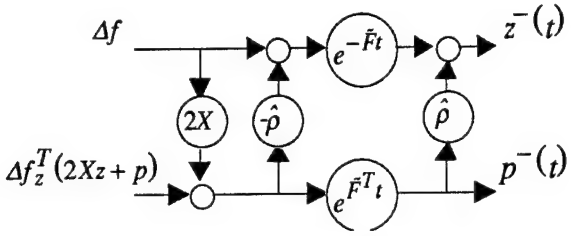
$$2Xz + p = \begin{bmatrix} 18 \\ 6 \end{bmatrix} e^{-t} + \begin{bmatrix} -8 \\ 0 \end{bmatrix} e^{-2t}$$

The next computation to be performed is the calculations of Δf and $\Delta f_z^T(2Xz+p)$ (denoted nonlinear calculations in Figure A>4). The specific operations are shown in Figure A.2 in the boxed region. The output Δf and $\Delta f_z^T(2Xz+p)$ (from Figure A.2) is

$$\Delta f = \begin{bmatrix}0.27\\0\end{bmatrix}e^{-2t} + \begin{bmatrix}-0.36\\0\end{bmatrix}e^{-3t} + \begin{bmatrix}0.12\\0\end{bmatrix}e^{-4t}$$

$$\Delta f_z^T(2Xz+p) = \begin{bmatrix}3.24\\0\end{bmatrix}e^{-2t} + \begin{bmatrix}-3.6\\0\end{bmatrix}e^{-3t} + \begin{bmatrix}0.96\\0\end{bmatrix}e^{-4t}$$

The next computations for this example are illustrated in the following Figure



The inputs are Δf and $\Delta f_z^T(2Xz+p)$ and the outputs are denoted $z^-(t)$ and $p^-(t)$, as illustrated in the figure. Performing these calculations yields:

$$z^-(t) = \begin{bmatrix}0.27\\-0.27\end{bmatrix} + \begin{bmatrix}0.66\\0.33\end{bmatrix}e^{-t} + \begin{bmatrix}-1.1\\-0.1\end{bmatrix}e^{-2t} + \begin{bmatrix}-0.45\\0.81\end{bmatrix}e^{-3t} + \begin{bmatrix}0.78\\-1.5\end{bmatrix}e^{-4t} + \begin{bmatrix}-0.07\\0.91\end{bmatrix}e^{-5t} + \begin{bmatrix}-0.06\\-0.18\end{bmatrix}e^{-6t}$$

$$p^-(t) = \begin{bmatrix}4.86\\4.86\end{bmatrix}e^{-3t} + \begin{bmatrix}-5.76\\-10.08\end{bmatrix}e^{-4t} + \begin{bmatrix}1.68\\6.72\end{bmatrix}e^{-5t} + \begin{bmatrix}0\\-1.44\end{bmatrix}e^{-6t}$$

As shown in Figure A.4, these $z^-(t)$ and $p^-(t)$ outputs feed the integration routines INT_0_T and INT_T_INF, respectively. The output from these integration routines is

$$z^+(t) = \begin{bmatrix}0.27\\-0.27\end{bmatrix}t + \begin{bmatrix}-0.66\\-0.33\end{bmatrix}e^{-t} + \begin{bmatrix}0.55\\0.05\end{bmatrix}e^{-2t} + \begin{bmatrix}0.15\\-0.27\end{bmatrix}e^{-3t} + \begin{bmatrix}-0.195\\0.375\end{bmatrix}e^{-4t} + \begin{bmatrix}0.014\\-0.182\end{bmatrix}e^{-5t} + \begin{bmatrix}0.01\\0.03\end{bmatrix}e^{-6t} + \begin{bmatrix}1.131\\2.327\end{bmatrix}e^{0}$$

$$p^+(t) = \begin{bmatrix}1.62\\1.62\end{bmatrix}e^{-3t} + \begin{bmatrix}-1.44\\-2.52\end{bmatrix}e^{-4t} + \begin{bmatrix}0.336\\1.344\end{bmatrix}e^{-5t} + \begin{bmatrix}0\\-0.24\end{bmatrix}e^{-6t}$$

To compute the nonlinear contribution to the control $p^+(t)$ is evaluated at $t=0$. This yields

$$\Delta V_x^T = p^+(t=0) = \begin{bmatrix}0.516\\0.204\end{bmatrix}$$

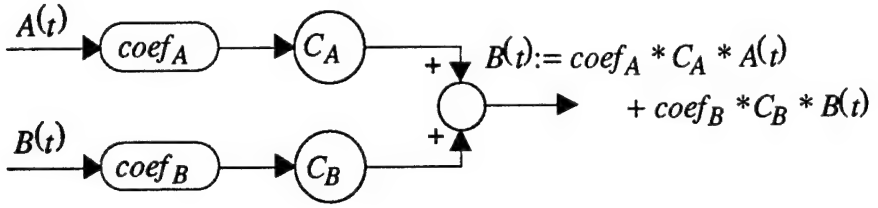


Figure A.8 Computational flow for subroutine *ADD2*

The contribution to the control is given by

$$u_{nonlinear} = -\tfrac{1}{2}R^{-1}B^T \Delta V_x^T = -\tfrac{1}{2}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^T\begin{bmatrix}0.516\\ 0.207\end{bmatrix} = 0.258$$

Program *NLHINF*

To run this program for your own application you must

1) First declare the following parameters

NSTATE = dimension of the state space
NR = dimension of the exogenous inputs
NM = dimension of the controls
NPMAX = maximum power of t allowed in any string coefficient
NAMAX = maximum number of exponential terms in any string

If either *NPMAX* or *NAMAX* are exceeded at any time during running of the program then an error message will be presented.

2) You must interface with a module which supplies the matrices $\tilde{A}, B, \tilde{Q}, \tilde{R}$.

3) you must specify the number of iterations desired.

4) The nonlinearities must be defined. Expressions must be derived for Δf and $\Delta f_z^T(2Xz + p)$, and these expressions must be programmed using the utility routines *MMULT*, *MULT*3, and *ADD*2. Any constant vectors or matrices needed for this must be calculated prior to the iteration loop. Any of the write statements found in the software can be used to aid program checkout or can be commented out.

Appendix B

Nonlinear H_∞ FORTRAN Software

This appendix contains a listing of the FORTRAN software program called NLHINF. This code can be obtained electronically by contacting the authors at wisek@mdcgwy.mdc.com.

```
      PROGRAM NLHINF
C
C ***********************************************************************
C     NONLINEAR H-INFINITY CONTROL                                      *
C     INPUTS ARE ATILDA,B,R AND QTILDA                                  *
C     FORMS RTILDA                                                      *
C     FORMS HAMILTONIAN MATRIX                                          *
C     FINDS THE EIGENVALUES AND EIGENVECTORS                            *
C     FORMS P MATRIX (SOLUTIONN OF LINEARIZED RICCATI EQUATION)         *
C      ATILDA'*P P*ATILDA + QTILDA  + P*B*RINVERSE*B'*P=0               *
C     FORMAS FTILDA MATRIX                                              *
C     FORMS EXP(FTILDA*t) AND RHOHAT                                    *
C     INCLUDES THE NONLINEAR CALCULATIONS AND                           *
C     PERFORMS THE ITERATIONS A PRESCRIBED NUMBER OF TIMES              *
C     CALCULATES THE NONLINEAR CONTRIBUTION TO THE CONTROL              *
C ***********************************************************************
C
      INTEGER I,J,K,L,M
      INTEGER NSTATES,NPMAX,NAMAX,NY,NR,NM
C
C     ---------------------------------------------------------
C
C     SET DIMENSIONS OF PROBLEM
      PARAMETER( NSTATES=2,NPMAX=20,NAMAX=3000)
      PARAMETER( NR=1) ! DIMENSION OF EXOGENOUS INPUTS
      PARAMETER( NM=1) ! DIMENSION OF CONTROLS
C
C     ---------------------------------------------------------
C
C     INPUTS
      REAL*8 X(NSTATES) ! STATEVECTOR MINUS TRIM STATEVECTOR
      REAL*8 ATILDA(NSTATES,NSTATES),B(NSTATES,NR+NM)
      REAL*8 R(NR+NM,NR+NM)
      REAL*8 QTILDA(NSTATES,NSTATES)
      INTEGER ITER ! DESIRED NUMBER OF ITERATIONS
C
C     COMPUTED QUANTITIES
      REAL*8 RTILDA(NSTATES,NSTATES)
      REAL*8 HAM(2*NSTATES,2*NSTATES),WRH(2*NSTATES),WIH(2*NSTATES)
      REAL*8 EIGVH(2*NSTATES,2*NSTATES)
      COMPLEX*8 EIG(NSTATES),Z1(NSTATES,NSTATES),Z2(NSTATES,NSTATES)
      REAL*8 TEMPH(2*NSTATES) ! TEMPORARY FOR EISPACK
      INTEGER ITEMP(2*NSTATES),IERR ! FOR EISPACK
      REAL*8 BT(NR+NM,NSTATES) ! FOR RTILDA CALCULATION
      REAL*8 RTEMP(NR+NM)     ! FOR RTILDA CALCULATION
      INTEGER INDXR(NR+NM),DD  !FOR RTILDA CALCULATION
      INTEGER INDX(NSTATES)  ! FOR FORMING P AND FTILDA
      COMPLEX*8 ATEMP(NSTATES,NSTATES),BTEMP(NSTATES) ! FOR LUDCMPC
      REAL*8 P(NSTATES,NSTATES)
      COMPLEX*8 Z1INV(NSTATES,NSTATES)
```

```
      REAL*8 FTILDA(NSTATES,NSTATES)
      REAL*8 RHOHAT(NSTATES,NSTATES)
C
C     EXPONENTIAL SUMS WITH POLYNOMIAL COEFFICIENTS
      COMPLEX*8 Z(NSTATES,NPMAX+1,NAMAX),ZEXP(NAMAX)
      COMPLEX*8 Q(NSTATES,NPMAX+1,NAMAX),QEXP(NAMAX)
      COMPLEX*8 TEMP(NSTATES,NPMAX+1,NAMAX),TEMPEXP(NAMAX)
      COMPLEX*8 DF(NSTATES,NPMAX+1,NAMAX),DFEXP(NAMAX)
      COMPLEX*8 FE(NSTATES,NSTATES,NSTATES+1,NSTATES),FEEXP(NSTATES)
      INTEGER NPZ,NAZ,NPQ,NAQ,NPTEMP,NATEMP,NPDF,NADF,NPFE,NAFE
C
C     INTERMEDIATE QUANTITIES
      INTEGER ITERCOUNT
      REAL*8 DVXT(NSTATES)
C
C     OUTPUTS
      REAL*8 Y(NR+NM) ! OUTPUT VECTOR = NONLINEAR CONTRIBUTION TO CONTROL
C
C     ---------------------------------------------------------------
C     DECLARATIONS FOR SPECIFIC NONLINEARITIES
      PARAMETER( C1=0.03)
      PARAMETER( PI=3.14159265)
      REAL*8 D,THETA
C
      REAL*8 E1(NSTATES),G(NSTATES,NSTATES),H(NSTATES,NSTATES)
C
C     PRELIMINARY PROBLEM DEPENDENT NONLINEAR CALCULATIONS
C
C     THE NONLINEARITY CONSIDERED IN THIS EXAMPLE IS
C
C     THE NONLINEARITY     |   2|
C                          | z1 |
C           deltaf(z)= c1*|     |
C                          |  0 |
C                          |    |
C     this requires that
C
C     deltaf= G*(E1'*z)*z    where G=[c1,0;0,0] , E1=[1;0],
C
C     deltafsubz'*w = H*(E1'*w)*z where H=[2*c1,0;0,0]
C

C     CALCULATE PRELIMINARY MATRICES
      E1(1)=1.
      E1(2)=0.
      G(1,1)=C1
      G(1,2)=0.
      G(2,1)=0.
      G(2,2)=0.
      H(1,1)=2*C1
      H(1,2)=0.
      H(2,1)=0.
      H(2,2)=0.
      WRITE(6,1)
      WRITE(6,*)15HTHE G MATRIX IS
100   FORMAT(' G(',I2,',',I2,')=',E25.17)
      DO I=1,NSTATES
       DO J=1,NSTATES
```

```
        WRITE(6,100) I,J,G(I,J)
       END DO
      END DO
      WRITE(6,1)
      WRITE(6,*)15HTHE H MATRIX IS
101   FORMAT(' H(',I2,',',I2,')=',E25.17)
      DO I=1,NSTATES
       DO J=1,NSTATES
        WRITE(6,101) I,J,H(I,J)
       END DO
      END DO
C
C     END OF PRELIMINARY MATRIX CALCULATION
C
C     ------------------------------------------------------
C
C     FORMATS FOR DIAGNOSTIC OUTPUTS
1     FORMAT('     ')
2     FORMAT('    NAZ=',I6,'       NPZ=',I3)
3     FORMAT('    NAQ=',I6,'       NPQ=',I3)
5     FORMAT(' DVXT(',I2,')=',E25.17)
6     FORMAT(' NATEMP=',I6,'    NPTEMP=',I3)
7     FORMAT(' BEGINNING OF ITERATION NUMBER ',I3)
8     FORMAT( I2)
9     FORMAT(' E1(1)  = ',E25.17,' E1(2) = ',E25.17)
C
C     SET DESIRED NUMBER OF ITERATIONS
      WRITE(6,*)34HENTER NUMBER OF ITERATIONS DESIRED
      READ(5,8) ITER
C     ITER=1
C
C     EXTRACT DATA FOR INPUT
C     (DUMMY VALUES ARE INSERTED FOR X,ATILDA,B,QTILDA AND R
C      FOR TEST CASE)
C     X(1) = 1.
C     X(2) = 2.
C
      WRITE(6,*) 11HENTER THETA
      READ(5,*) THETA
      WRITE(6,*) THETA
      WRITE(6,*) 7HENTER D
      READ(5,*) D
      WRITE(6,*) D
      X(1)=D*COS(THETA*PI/180.0)
      X(2)=D*SIN(THETA*PI/180.0)
C
C
C
      DO I=1,NSTATES
       DO J=1,NSTATES
          ATILDA(I,J)=0.
          RTILDA(I,J)=0.
          FTILDA(I,J)=0.
       END DO
       DO J=1,NR+NM
        B(I,J)=0.
       END DO
      END DO
```

```
      DO I=1,NR+NM
       DO J=1,NR+NM
        R(I,J)=0.
       END DO
      END DO

      ATILDA(1,1) = -4.
      ATILDA(1,2) = 0.
      ATILDA(2,1) = 1.
      ATILDA(2,2) = -1.

      B(1,1) = 1.
      B(2,2) = 1.

      R(1,1) = -1.
      R(2,2) = 1.
C
      QTILDA(1,1)=14.
      QTILDA(1,2)=2.
      QTILDA(2,1)=2.
      QTILDA(2,2)=2.
C
10    FORMAT(' R(',I2,',',I2,')=',E25.17)
C     DO I=1,NR+NM
C      DO J=1,NR+NM
C       WRITE(6,10) I,J,R(I,J)
C      END DO
C     END DO
C
11    FORMAT(' B(',I2,',',I2,')=',E25.17)
C     DO I=1,NSTATES
C      DO J=1,NR+NM
C       WRITE(6,11) I,J,B(I,J)
C      END DO
C     END DO
C
C     ************************************************************
C     FORM RTILDA MATRIX
C
C     *******************************************
C     *  RTILDA=-B*RINVERSE*BTRANSPOSED         *
C     *******************************************
C
      CALL LUDCMPR(R,NR+NM,NR+NM,INDXR,DD)
      DO I=1,NR+NM
       DO J=1,NSTATES
        BT(I,J)=B(J,I)
       END DO
      END DO
      DO I=1,NSTATES
       DO J=1,NR+NM
        RTEMP(J)=BT(J,I)
       END DO
       CALL LUBKSBR(R,NR+NM,NR+NM,INDXR,RTEMP)
       DO J=1,NSTATES
        RTILDA(I,J)=0.
        DO K=1,NR+NM
         RTILDA(I,J)=RTILDA(I,J)-B(J,K)*RTEMP(K)
```

```
        END DO
       END DO
      END DO
C
12    FORMAT(' RTILDA(',I2,',',I2,')=',E25.17)
C     DO I=1,NSTATES
C      DO J=1,NSTATES
C       WRITE(6,12) I,J,RTILDA(I,J)
C      END DO
C     END DO
C
C     ****************************************************
C
C     FORM HAMILTONIAN MATRIX
C
C     *********************************
C     *                               *
C     *      | ATILDA     RTILDA    | *
C     *  H = |                      | *
C     *      | -QTILDA    -ATILDA'  | *
C     *                               *
C     *********************************
C
      DO I=1,NSTATES
       DO J=1,NSTATES
        HAM(I,J)=ATILDA(I,J)
        HAM(I,J+NSTATES)=RTILDA(I,J)
        HAM(I+NSTATES,J)=-QTILDA(I,J)
        HAM(I+NSTATES,J+NSTATES)=-ATILDA(J,I)
       END DO
      END DO
C
13    FORMAT(' HAM(',I2,',',I2,')=',E25.17)
C     WRITE(6,1)
C     DO I=1,2*NSTATES
C      DO J=1,2*NSTATES
C       WRITE(6,13) I,J,HAM(I,J)
C      END DO
C      WRITE(6,1)
C     END DO
C
C     SOLVE FOR EIGENVALUES AND EIGENVECTORS OF HAMILTONIAN
      CALL ZZ_RG(2*NSTATES,2*NSTATES,HAM,WRH,WIH,1,
     *               EIGVH,ITEMP,TEMPH,IERR)
C
C
14    FORMAT(' EIGH(',I2,')=',E25.17,',',E25.17)
C     WRITE(6,1)
C     WRITE(6,*)34HEIGENVALUES OF THE HAMILTONIAN ARE
C     DO I=1,2*NSTATES
C      WRITE(6,14) I,WRH(I),WIH(I)
C     END DO
C
C
15    FORMAT(' EIGVH(',I2,',',I2,')=',E25.17)
C     WRITE(6,1)
C     DO J=1,2*NSTATES
C      DO I=1,2*NSTATES
```

```
C       WRITE(6,15) I,J,EIGVH(I,J)
C      END DO
C      WRITE(6,1)
C     END DO
C
C     ************************************************************
C
C     SELECT STABLE EIGENVALUES AND CORRESPONDING EIGENVECTORS
C
C     ********************************************
C     *                                          *
C     *           | Z1 |    | Z1 |                *
C     *       H * |    | =  |    | * DIAG( EIGH)  *
C     *           | Z2 |    | Z2 |                *
C     *                                          *
C     ********************************************
C
C     BUILD Z1 AND Z2 MATRICES
      K=0
      DO I=1,2*NSTATES
       IF(WRH(I).LT.0.) THEN
        K=K+1
        EIG(K)=CMPLX(WRH(I),WIH(I))
        IF(WIH(I).EQ.0.) THEN
         DO J=1,NSTATES
          Z1(J,K)=EIGVH(J,I)
          Z2(J,K)=EIGVH(J+NSTATES,I)
         END DO
        END IF
        IF(WIH(I).GT.0.) THEN
         DO J=1,NSTATES
          Z1(J,K)=CMPLX(EIGVH(J,I),EIGVH(J,I+1))
          Z2(J,K)=CMPLX(EIGVH(J+NSTATES,I),EIGVH(J+NSTATES,I+1))
         END DO
        END IF
        IF(WIH(I).LT.0.) THEN
         DO J=1,NSTATES
          Z1(J,K)=CMPLX(EIGVH(J,I-1),-EIGVH(J,I))
          Z2(J,K)=CMPLX(EIGVH(J+NSTATES,I-1),-EIGVH(J+NSTATES,I))
         END DO
        END IF
       END IF
      END DO
C
16    FORMAT(' Z1(',I2,',',I2,')=',E25.17,',',E25.17)
C     WRITE(6,1)
C     DO J=1,NSTATES
C      DO I=1,NSTATES
C       WRITE(6,16) I,J,REAL(Z1(I,J)),IMAG(Z1(I,J))
C      END DO
C      WRITE(6,1)
C     END DO
C
17    FORMAT(' Z2(',I2,',',I2,')=',E25.17,',',E25.17)
C     WRITE(6,1)
C     DO J=1,NSTATES
C      DO I=1,NSTATES
C       WRITE(6,17) I,J,REAL(Z2(I,J)),IMAG(Z2(I,J))
```

```
C       END DO
C       WRITE(6,1)
C      END DO
C
C
18     FORMAT(' EIG(',I2,')=',E25.17,',',E25.17)
C      WRITE(6,1)
C      DO I=1,NSTATES
C       WRITE(6,18) I,REAL(EIG(I)),IMAG(EIG(I))
C      END DO
C
C      ************************************************
C
C      FORM SOLUTION TO THE ALGEBRAIC RICCATI EQUATION
C
C      ***************************
C      *    P=Z2*Z1INVERSE       *
C      ***************************
C
       DO I=1,NSTATES
        DO J=1,NSTATES
         ATEMP(I,J)=Z1(J,I)
        END DO
       END DO
       CALL LUDCMPC(ATEMP,NSTATES,NSTATES,INDX,DD)
       DO I=1,NSTATES
        DO J=1,NSTATES
         BTEMP(J)=Z2(I,J)
        END DO
        CALL LUBKSBC(ATEMP,NSTATES,NSTATES,INDX,BTEMP)
        DO J=1,NSTATES
         P(I,J)=REAL(BTEMP(J))
        END DO
       END DO
C
C
       DO I=1,NSTATES
        DO J=1,NSTATES
         BTEMP(J)=0.
        END DO
        BTEMP(I)=1.
        CALL LUBKSBC(ATEMP,NSTATES,NSTATES,INDX,BTEMP)
        DO J=1,NSTATES
         Z1INV(I,J)=BTEMP(J)
        END DO
       END DO
C
C
19     FORMAT(' P(',I2,','I2,')=',E25.17)
       DO I=1,NSTATES
        DO J=1,NSTATES
         WRITE(6,19)I,J,P(I,J)
        END DO
       END DO
       CALL TESTP(ATILDA,QTILDA,RTILDA,P,NSTATES)
C
C      ************************************************
C      FORM FTILDA MATRIX
```

```
C
C     ***********************************
C     *  FTILDA=Z1*DIAG*EIG)*Z1INVERSE  *
C     ***********************************
C
      DO I=1,NSTATES
       DO J=1,NSTATES
        BTEMP(J)=EIG(J)*Z1(I,J)
       END DO
       CALL LUBKSBC(ATEMP,NSTATES,NSTATES,INDX,BTEMP)
       DO J=1,NSTATES
        FTILDA(I,J)=BTEMP(J)
       END DO
      END DO
C
20    FORMAT(' FTILDA(',I2,',',I2,')=',E25.17)
      DO I=1,NSTATES
       DO J=1,NSTATES
        WRITE(6,20) I,J,FTILDA(I,J)
       END DO
      END DO
      CALL TESTF(ATILDA,RTILDA,P,FTILDA,NSTATES)
C
C     ************************************************************
C     FORM RHOHAT MATRIX
C
C     ************************************************
C     *                                              *
C     *   FTILDA*RHOHAT + RHOHAT*FTILDA' = RTILDA/2  *
C     *                                              *
C     ************************************************
C
      CALL RHAT(FTILDA,RTILDA,RHOHAT,NSTATES)
21    FORMAT(' RHOHAT(',I2,',',I2,')=',E25.17)
      DO I=1,NSTATES
       DO J=1,NSTATES
        WRITE(6,21) I,J,RHOHAT(I,J)
       END DO
      END DO
C
C     FORM MATRIX EXPONENTIAL
      NPFE=0
      NAFE=NSTATES
      DO I=1,NSTATES
       DO J=1,NSTATES
        DO L=1,NSTATES
         FE(I,J,1,L)=Z1(I,L)*Z1INV(L,J)
        END DO
       END DO
       FEEXP(I)=EIG(I)
      END DO
C
C
C     ********************************************************
C     FORM MATRIX EXPONENTIAL
C
C     CALL MATEXP(FTILDA,EIG,FE,FEEXP,NPFE,NAFE,NSTATES,NSTATES)
C
```

```
      CALL WRITEXP(FE,FEEXP,NPFE,NAFE,NSTATES)
C
C     ********************************************************
C
C
C     INITIALIZE Z,ZEXP,Q,QEXP
C
      NPZ=0
      NAZ=1
      DO L=1,NAZ
       ZEXP(L)=0.
       DO I=1,NSTATES
        DO K=1,NPZ+1
         Z(I,K,L)=X(I)
        END DO
       END DO
      END DO
C
      NPQ=0
      NAQ=1
      DO L=1,NAQ
       QEXP(L)=0.
       DO I=1,NSTATES
        DO K=1,NPQ+1
         Q(I,K,L)=0.
        END DO
       END DO
      END DO
      WRITE(6,1)
      WRITE(6,*)20HAFTER INITIALIZATION
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,
     *                  NSTATES,NPMAX,NAMAX)
C
C
C     START ITERATIONS ***************************************
C
      DO ITERCOUNT=1,ITER
       WRITE(6,7) ITERCOUNT
       WRITE(6,2) NAZ,NPZ
      WRITE(6,1)
      WRITE(6,*)25HAT BEGINNING OF ITERATION
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL ADD(Q,QEXP,NPQ,NAQ,Z,ZEXP,NPZ,NAZ,1.,RHOHAT,
     *                                  NSTATES,NPMAX,NAMAX)
      WRITE(6,*)15HAFTER FIRST ADD

       WRITE(6,2) NAZ,NPZ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL MULT(FE,FEEXP,NPFE,NAFE,Z,ZEXP,NPZ,NAZ,TEMP,TEMPEXP,
     *                                  NSTATES,NPMAX,NAMAX,0,0)
      WRITE(6,*)16HAFTER FIRST MULT
       WRITE(6,2) NAZ,NPZ
C
```

```
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL MULT(FE,FEEXP,NPFE,NAFE,Q,QEXP,NPQ,NAQ,TEMP,TEMPEXP,
     *                                    NSTATES,NPMAX,NAMAX,1,1)
      WRITE(6,*)11HSECOND MULT
       WRITE(6,3) NAQ,NPQ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL ADD(Q,QEXP,NPQ,NAQ,Z,ZEXP,NPZ,NAZ,-1.,RHOHAT,
     *                                    NSTATES,NPMAX,NAMAX)
      WRITE(6,*)16HAFTER SECOND ADD
       WRITE(6,2) NAZ,NPZ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL ADD(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,2.,P,
     *           NSTATES,NPMAX,NAMAX)
          WRITE(6,*)20HAFTER THIRD ADD ****
          WRITE(6,3) NAQ,NPQ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C     ----------------------------------------------------------------
C
C         NONLINEAR PROBLEM DEPENDENT PART OF ITERATION*****************
C
C         CALCULATE DELTA F
C
          WRITE(6,9)E1(1),E1(2)
          CALL MULT3(E1,Z,ZEXP,NPZ,NAZ,Z,ZEXP,NPZ,NAZ,
     *                 DF,DFEXP,NPDF,NADF,NSTATES,NPMAX,NAMAX)
          WRITE(6,*)10HFIRSTMULT3
C
      CALL WRITE_ZQ(DF,DFEXP,NPDF,NADF,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C

          CALL TRIM_NA(DF,DFEXP,NPDF,NADF,NSTATES,NPMAX,NAMAX,0.0000001)
          WRITE(6,*)24HTRIMNA AFTER FIRST MULT3
C
      CALL WRITE_ZQ(DF,DFEXP,NPDF,NADF,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL MMULT(G,DF,DFEXP,NPDF,NADF,NPMAX,NAMAX,NSTATES)
          WRITE(6,*)11HFIRST MMULT
          WRITE(6,6) NADF,NPDF
C
      CALL WRITE_ZQ(DF,DFEXP,NPDF,NADF,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
C         CALCULATE MINUS Q
C
          CALL MULT3(E1,Q,QEXP,NPQ,NAQ,Z,ZEXP,NPZ,NAZ,
     *                 TEMP,TEMPEXP,NPTEMP,NATEMP,NSTATES,NPMAX,NAMAX)
          WRITE(6,*)18HAFTER SECOND MULT3
          WRITE(6,6) NATEMP,NPTEMP
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,TEMP,TEMPEXP,NPTEMP,NATEMP,
     *                   NSTATES,NPMAX,NAMAX)
C
          CALL MMULT(H,TEMP,TEMPEXP,NPTEMP,NATEMP,NPMAX,NAMAX,NSTATES)
```

```
          WRITE(6,*)18HAFTER SECOND MMULT
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,TEMP,TEMPEXP,NPTEMP,NATEMP,
     *                    NSTATES,NPMAX,NAMAX)
C
          CALL COPY(TEMP,TEMPEXP,NPTEMP,NATEMP,Q,QEXP,NPQ,NAQ,
     *                      NSTATES,NPMAX,NAMAX)
          CALL COPY(DF,DFEXP,NPDF,NADF,Z,ZEXP,NPZ,NAZ,
     *                      NSTATES,NPMAX,NAMAX)
          WRITE(6,*)10HAFTER COPY
          WRITE(6,3) NAQ,NPQ
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
C         END OF PROBLEM DEPENDENT NONLINEAR CALCULATIONS**************
C
C      ---------------------------------------------------------------
C
          CALL ADD(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,2.,P,
     *            NSTATES,NPMAX,NAMAX)
          WRITE(6,*)16HAFTER FOURTH ADD
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL TRIM_NA(Z,ZEXP,NPZ,NAZ,NSTATES,NPMAX,NAMAX,0.000001)
          WRITE(6,*)18HAFTER FIRST TRIMNA
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL TRIM_NA(Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX,0.000001)
          WRITE(6,*)19HAFTER SECOND TRIMNA
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
C       NON-PROBLEM DEPENDENT CALCULATIONS
C
          CALL ADD(Q,QEXP,NPQ,NAQ,Z,ZEXP,NPZ,NAZ,-1.,RHOHAT,
     *                                NSTATES,NPMAX,NAMAX)
       WRITE(6,*)15HAFTER FIFTH ADD
          WRITE(6,2) NAZ,NPZ
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL MULT(FE,FEEXP,NPFE,NAFE,Z,ZEXP,NPZ,NAZ,TEMP,TEMPEXP,
     *                                NSTATES,NPMAX,NAMAX,0,1)
       WRITE(6,*)16HAFTER THIRD MULT
          WRITE(6,2) NAZ,NPZ
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL MULT(FE,FEEXP,NPFE,NAFE,Q,QEXP,NPQ,NAQ,TEMP,TEMPEXP,
     *                                NSTATES,NPMAX,NAMAX,1,0)
       WRITE(6,*)17HAFTER FOURTH MULT
          WRITE(6,3) NAQ,NPQ
C
       CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
          CALL ADD(Q,QEXP,NPQ,NAQ,Z,ZEXP,NPZ,NAZ,1.,RHOHAT,
```

```
     *                                        NSTATES,NPMAX,NAMAX)
      WRITE(6,*)15HAFTER SIXTH ADD
       WRITE(6,2) NAZ,NPZ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL INT_0_T(X,Z,ZEXP,NPZ,NAZ,NSTATES,NPMAX,NAMAX)
      WRITE(6,*)15HAFTER INT0 TO T
       WRITE(6,2) NAZ,NPZ
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL INT_T_INF(Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
      WRITE(6,*)17HAFTER INTT TO INF
       WRITE(6,3) NAQ,NPQ
C
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALL TRIM_NA(Z,ZEXP,NPZ,NAZ,NSTATES,NPMAX,NAMAX,0.000001)
      WRITE(6,*)18HAFTER THIRD TRIMNA
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
       CALLTRIM_NA(Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX,0.000001)
      WRITE(6,*)20HAFTER FOURTH TRIMNA*
      END DO !! END OF ITERATIONS ******************************
C
C
      CALL WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,NPMAX,NAMAX)
C
      CALL ZEROEVAL(Q,NPQ,NAQ,DVXT,NSTATES,NPMAX,NAMAX)
      DO I=1,NSTATES
       WRITE(6,5) I,DVXT(I)
      END DO
C
C     ************************************************************
C     COMPUTE NONLINEAR CONTRIBUTION TO THE CONTROL
C     ****************************************
C     * Y=-0.5*RINVERSE*BTRANSPOSE*DVDXT *
C     ****************************************
      DO I=1,NR+NM
       Y(I)=0.
       DO J=1,NSTATES
        Y(I)=Y(I)-BT(I,J)*DVXT(J)/2.
       END DO
      END DO
      CALL LUBKSBR(R,NR+NM,NR+NM,INDXR,Y)
C
      WRITE(6,1)
      WRITE(6,*)37HNONLINEAR CONTRIBUTION TO THE CONTROL
22    FORMAT(' Y(',I2,')=',E24.17)
      DO I=1,NM
       WRITE(6,22) I,Y(I)
      END DO
C     STOP
      END   !NLHINF
C     ************************************************************
```

```
C     ************************************************************
      SUBROUTINE RHAT(F,RTILDA,RHOHAT,NSTATES)
C
C     ***********************************************
C     *                                             *
C     *   F*RHOHAT + RHOHAT*F' = RTILDA/2           *
C     *                                             *
C     ***********************************************
C
      INTEGER NSTATES,NMAX,NNMAX
      PARAMETER( NMAX=20)
      PARAMETER( NNMAX=(NMAX*(NMAX+1))/2)
      INTEGER NN,INDX(NNMAX)
      REAL*8 F(NSTATES,NSTATES),RTILDA(NSTATES,NSTATES)
      REAL*8 RHOHAT(NSTATES,NSTATES)
      REAL*8 R(NMAX,NMAX)
      REAL*8 FF(NNMAX,NNMAX),RR(NNMAX)
      REAL*8 D
      INTEGER I,J
C
      NN=(NSTATES*(NSTATES+1))/2
C     R=RTILDA/2
C
      DO I=1,NSTATES
       DO J=1,NSTATES
        R(I,J)=RTILDA(I,J)/2.
       END DO
      END DO
      CALL STACK2(F,FF,NSTATES,NN,NNMAX)
      CALL STACK1(R,RR,NSTATES,NN,NMAX,NNMAX)
      CALL LUDCMPR(FF,NN,NNMAX,INDX,D)
      CALL LUBKSBR(FF,NN,NNMAX,INDX,RR)
      CALL UNSTACK(RR,RHOHAT,NSTATES,NN)
      RETURN
      END  !RHAT
C     ************************************************************
C       ************************************************************
      SUBROUTINE UNSTACK(XX,X,N,NN)
C     *****************************************
C     *    INVERSE OF SUBROUTINE STACK1       *
C     *****************************************
      REAL*8 X(N,N),XX(NN)
      INTEGER I,J,N,NN,M
C
      M=1
      DO I=1,N
       DO J=I,N
        X(I,J)=XX(M)
        X(J,I)=XX(M)
        M=M+1
       END DO
      END DO
      RETURN
      END   !UNSTACK
C     ************************************************************
C     ************************************************************
      SUBROUTINE STACK1(X,XX,N,NN,NMAX,NNMAX)     ! NN=(N*(N+1))/2
C     ***********************************************************
```

```
C     * X is an N by N symmetric matrix                          *
C     * XX is an NN by 1 stacked version of X proceeding along *
C     * portions of rows on and above the diagonal               *
C     ***********************************************************
      REAL*8 X(NMAX,NMAX),XX(NNMAX)
      INTEGER I,J,II
C
      II=1
      DO I=1,N
       DO J=I,N
        XX(II)=X(I,J)
        II=II+1
       END DO
      END DO
      RETURN
      END !STACK1
C     ***************************************************************
C     ***************************************************************
      SUBROUTINE STACK2(A,AA,N,NN,NNMAX)     ! NN=(N*(N+1))/2
C     ***********************************************************
C     * Changes the A matrix in A*X+X*Atranspose=B (with X and B *
C     * symmetric) into the AA in AA*XX=BB where XX and BB are   *
C     * stacked versions of X and B respectively                 *
C     ***********************************************************
      REAL*8 A(N,N), AA(NNMAX,NNMAX)
      INTEGER I,J,K,II,JJ,N,NN
C
C
      DO II=1,NN
       DO JJ=1,NN
        AA(II,JJ)=0.0
       END DO
      END DO
C
      II=1
      DO I=1,N
       DO J=I,N
        DO K=1,N
         CALL STACK(N,K,J,JJ)
         AA(II,JJ)=AA(II,JJ)+A(I,K)
         CALL STACK(N,K,I,JJ)
         AA(II,JJ)=AA(II,JJ)+A(J,K)
        END DO
        II=II+1
       END DO
      END DO
C
      RETURN
      END   !STACK2
C     ***************************************************************
C     ***************************************************************
      SUBROUTINE STACK(N,I,J,II)
C     ***********************************************************
C     * TRANSFORMS ROW AND COLUMN INDICES OF N BY N SYMMETRIC *
C     * MATRIX TO INDEX OF THE STACKED MATRIX                    *
C     ***********************************************************
      INTEGER N,I,J,II
      IF(J.GE.I) II=((2*N-I)*(I-1))/2+J
```

```
        IF(J.LT.I) II=((2*N-J)*(J-1))/2+I
        RETURN
      END   !STACK
C     ****************************************************************
C     ****************************************************************
      SUBROUTINE LUDCMPR(A,N,NP,INDX,D)
c     Given an nxn matrix a, with physical dimension np, this
c     routine replaces it by the rowise permutation of itself.
c     A and N are input, A is output. INDX is an output vector
c     which records the row permutation effected by the partial
c     pivoting; D is output as +1 or -1 depending on whether the
c     number of row interchanges was even or odd, respectively.
c     This routine is used in combination with LUBKSBR to solve
c     linear equations or invert a matrix.
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
c
1     FORMAT(' Singular matrix in LUDCMPR')
      PARAMETER (NMAX=100,TINY=1.0e-20)
      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      D=1.0
C
      DO I=1,N
       AAMAX=0.0
       DO J=1,N
        IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
       END DO
        IF (AAMAX.EQ.0.0) WRITE(6,1)    !singular matrix
        IF (AAMAX.EQ.0.0) PAUSE         !singular matrix
       VV(I)=1.0/AAMAX
      END DO
      DO J=1,N
       IF (J.GT.1) THEN
        DO I=1,J-1
         SUM=A(I,J)
         IF (I.GT.1) THEN
          DO K=1,I-1
           SUM=SUM-A(I,K)*A(K,J)
          END DO
          A(I,J)=SUM
         END IF
        END DO
       END IF
       AAMAX=0.0
       DO I=J,N
        SUM=A(I,J)
        IF (J.GT.1) THEN
         DO K=1,J-1
          SUM=SUM-A(I,K)*A(K,J)
         END DO
         A(I,J)=SUM
        END IF
        DUM=VV(I)*ABS(SUM)
        IF (DUM.GE.AAMAX) THEN
         IMAX=I
         AAMAX=DUM
        END IF
       END DO
       IF (J.NE.IMAX) THEN
```

```
          DO K=1,N
           DUM=A(IMAX,K)
           A(IMAX,K)=A(J,K)
           A(J,K)=DUM
          END DO
          D=-D
          VV(IMAX)=VV(J)
         END IF
         INDX(J)=IMAX
         IF (J.NE.N) THEN
          IF (A(J,J).EQ.0.0) A(J,J)=TINY
          DUM=1.0/A(J,J)
          DO I=J+1,N
           A(I,J)=A(I,J)*DUM
          END DO
         END IF
        END DO
       IF (A(N,N).EQ.0.0) A(N,N)=TINY
       RETURN
       END     !LUDCMPR
C      ****************************************************************
C      ****************************************************************
       SUBROUTINE LUBKSBR(A,N,NP,INDX,B)
C      Solves the set of N linear equations A*X=B. Here A is input
C      not as the matrix A but rather as its LU decomposition,
C      determined by the routine LUDCMPR. B is input as the right-hand
C      side vector B, and returns with the solution vector X. A,N,NP
C      and INDX are not modified by this routine and can be left in
C      place for successive calls with different right-hand sides B.
C      This routine takes into account the possibility that B will
C      begin with many zero elements, so it is sofficient for use in
C      matrix inversion
       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
       DIMENSION A(NP,NP),INDX(N),B(N)
       II=0
       DO I=1,N
        LL=INDX(I)
        SUM=B(LL)
        B(LL)=B(I)
        IF (II.NE.0) THEN
         DO J=II,I-1
          SUM=SUM-A(I,J)*B(J)
         END DO
        ELSE IF (SUM.NE.0) THEN
         II=I
        END IF
        B(I)=SUM
       END DO
       DO I=N,1,-1
        SUM=B(I)
        IF (I.LT.N) THEN
         DO J=I+1,N
          SUM=SUM-A(I,J)*B(J)
         END DO
        END IF
        B(I)=SUM/A(I,I)
       END DO
       RETURN
```

```
      END   !LUBKSBR
C     ************************************************************
C     ************************************************************
      SUBROUTINE MATEXP(A,EIG,AE,AEEXP,NP,NA,N,NMAX)
C     ************************************************************
C     * INPUT N,NMAX,A(NMAX,NMAX),EIG(NMAX)                      *
C     * RETURNS NP,NA,AE(N,N,N+1,N),AEEXP(N)                     *
C     * AE REPRESENTS THE MATRIX EXPONENTIAL OF A                *
C     * [e^(A*t)](i,j)=sum on k and l of                         *
C     * AE(i,j,k,l)*t^(k-1)*e^(AEEXP(l)*t)                       *
C     ************************************************************
      INTEGER NMAX,NMAXX
      PARAMETER( NMAXX=20)
      INTEGER N,NP,NA,INDX(NMAXX),NEIG(NMAXX)
      REAL*8 A(NMAX,NMAX),ATEMP(NMAXX,NMAXX)
      REAL*8 WR(NMAXX),WI(NMAXX)
      COMPLEX*8 EIG(NMAX)
      COMPLEX*8 AE(NMAX,NMAX,NMAX+1,NMAX),AEEXP(NMAX)
      COMPLEX*8 VDM(NMAXX,NMAXX),VDMINV(NMAXX,NMAXX)
      REAL*8 G(NMAXX,NMAXX),GTEMP(NMAXX,NMAXX)
      INTEGER I,J,K,L,M,NCOEF,KK,LL
      INTEGER INDX1(NMAXX) !FOR MATRIX INVERSION ROUTINES
      REAL D !FOR MATRIX INVERSION ROUTINES
      LOGICAL FLAG
      COMPLEX*8 ZERO
C
C     THE FOLLOWING ARE DIMENSION STATEMENTS FOR A DIAGNOSTIC
      REAL*8 T,TIME(4),AEVALR(NMAXX,NMAXX),AER,AEI,EIGR,EIGI
      REAL*8 AEVALI(NMAXX,NMAXX)
C
      INTEGER IERR,ITEMP(NMAXX)   !FOR EIGENVALUE ROUTINE ZZ_RG
      REAL*8 TEMP2(NMAXX),Z(NMAXX,NMAXX)   !FOR EIGENVALUE ROUTINE ZZ_RG
C
C     FIND EIGENVALUES OF A
C     DO I=1,N
C      DO J=1,N
C       ATEMP(I,J)=A(I,J)
C      END DO
C     END DO
C
C     CALL ZZ_RG(NMAXX,N,ATEMP,WR,WI,0,Z,ITEMP,TEMP2,IERR)
C     DO I=1,N
C        EIG(I)=CMPLX(WR(I),WI(I))
C     END DO
1     FORMAT(' EIG(',I2,')=',E25.17,',',E25.17)
C     DO I=1,N
C      WRITE(6,1) I,WR(I),WI(I)
C     END DO
C
C
C
      CALL EIGSRT(N,EIG)
C
2     FORMAT('  FTILDA MATRIX IS NOT STABLE')
      IF(REAL(EIG(1)).GE.0.)THEN
       WRITE(6,2)
       PAUSE
      END IF
```

```
C
C       FORM INDEX VECTOR TO REPRESENT MULTIPLICITIES
        INDX(1)=0
        DO I=2,N
         IF(EIG(I).EQ.EIG(I-1)) THEN
          INDX(I)=INDX(I-1)+1
         ELSE
          INDX(I)=0
         END IF
        END DO
C
C       FORM VANDERMONDE MATRIX
        DO I=1,N
         DO J=1,INDX(I)
          VDM(I,J)=(0.,0.)
         END DO
         DO J=INDX(I)+1,N
          L=J-1
          NCOEF=1
          DO K=1,INDX(I)
           NCOEF=NCOEF*L
           L=L-1
          END DO
          VDM(I,J)=NCOEF*(EIG(I)**L)
         END DO
        END DO
C
C       FIND NUMBER OF EXPONENTIAL TERMS AND ORDER OF
C       POLYNOMIAL COEFICIENTS
C
        NEIG(1)=1
        DO I=2,N
         IF(EIG(I).EQ.EIG(I-1)) THEN
          NEIG(I)=NEIG(I-1)
         ELSE
          NEIG(I)=NEIG(I-1)+1
         END IF
        END DO
        NA=NEIG(N)
C
        NP=0
        DO I=1,N
         IF(INDX(I).GT.NP) THEN
          NP=INDX(I)
         END IF
        END DO
C
C       INVERT VANDERMONDE MATRIX
C
        DO I=1,N
         DO J=1,N
          VDMINV(I,J)=(0.,0.)
         END DO
         VDMINV(I,I)=(1.,0.)
        END DO
        CALL LUDCMPC(VDM,N,NMAXX,INDX1,D)
C
        DO J=1,N
```

```
        CALL LUBKSBC(VDM,N,NMAXX,INDX1,VDMINV(1,J))
       END DO
C
C      SET G=IDENTITY MATRIX
       DO L=1,N
        DO M=1,N
         G(L,M)=0.
        END DO
        G(L,L)=1.
       END DO
C
C      CALCULATE THE INTERPOLATING MATRICES
       DO K=1,NP+1
        DO L=1,NA
         DO I=1,N
          DO J=1,N
           AE(I,J,K,L)=(0.,0.)
          END DO
         END DO
        END DO
       END DO
       DO K=1,N
        DO L=1,N
         DO I=1,N
          DO J=1,N
           KK=INDX(L)+1
           LL=NEIG(L)
           AE(I,J,KK,LL)=AE(I,J,KK,LL)+G(I,J)*VDMINV(K,L)
          END DO
         END DO
        END DO
C
C       INCREASE POWER OF A SO THAT G=A*G=A^K FOR NEXT K
        DO I=1,N
         DO J=1,N
          GTEMP(I,J)=0.
          DO L=1,N
           GTEMP(I,J)=GTEMP(I,J)+A(I,L)*G(L,J)
          END DO
         END DO
        END DO
        DO I=1,N
         DO J=1,N
          G(I,J)=GTEMP(I,J)
         END DO
        END DO
C
       END DO
C
       AEEXP(1)=EIG(1)
       J=1
       DO I=2,N
        IF(NEIG(I).NE.NEIG(I-1)) THEN
         J=J+1
         AEEXP(J)=EIG(I)
        END IF
       END DO
C
```

```
C     TRIM MAX POWER OF t IF NECESSARY
      ZERO=(0.,0.)
      DO K=NP+1,2,-1
       FLAG=.TRUE.
       DO L=1,NA
        DO I=1,N
         DO J=1,N
          IF (AE(I,J,K,L).NE.ZERO) THEN
           FLAG=.FALSE.
          END IF
         END DO
        END DO
       END DO
       IF(FLAG) THEN
        NP=NP-1
       END IF
      END DO
C
C     **************DIAGNOSTIC EVALUATION*****************
3     FORMAT(' AEXP(',I2,',',I2,')=',E25.17,',',E25.17)
4     FORMAT(' e**(A*',E 25.17,') IS')
5     FORMAT('  ')
      TIME(1)=0.
      TIME(2)=0.01
      TIME(3)=0.02
      TIME(4)=0.03
      DO M=1,4
       T=TIME(M)
       WRITE(6,5)
       WRITE(6,4) T
       DO I=1,N
        DO J=1,N
         AEVALR(I,J)=0.
         AEVALI(I,J)=0.
         DO K=1,NP+1
          DO L=1,NA
           AER=REAL(AE(I,J,K,L))
           AEI=IMAG(AE(I,J,K,L))
           EIGR=REAL(EIG(L))
           EIGI=IMAG(EIG(L))
           IF(K.NE.1) THEN
            AEVALR(I,J)=AEVALR(I,J)+(T**(K-1))*EXP(EIGR*T)
     *                        *(AER*COS(EIGI*T)-AEI*SIN(EIGI*T))
            AEVALI(I,J)=AEVALI(I,J)+(T**(K-1))*EXP(EIGR*T)
     *                        *(AEI*COS(EIGI*T)+AER*SIN(EIGI*T))
           ELSE
            AEVALR(I,J)=AEVALR(I,J)+EXP(EIGR*T)
     *                        *(AER*COS(EIGI*T)-AEI*SIN(EIGI*T))
            AEVALI(I,J)=AEVALI(I,J)+EXP(EIGR*T)
     *                        *(AEI*COS(EIGI*T)+AER*SIN(EIGI*T))
           END IF
          END DO
         END DO
         WRITE(6,3)I,J,AEVALR(I,J),AEVALI(I,J)
        END DO
       END DO
      END DO
C     *************END OF DIAGNOSTIC*******************
```

```
C
C
      RETURN
      END   !MATEXP
C     ***************************************************************
C     ***************************************************************
      SUBROUTINE LUDCMPC(A,N,NP,INDX,D)
C     ****************************************
C     * COMPLEX VERSION OF LU-DECOMPOSITION  *
C     ****************************************
      INTEGER N,NP,NMAX
      PARAMETER(NMAX=100,TINY=1.0E-20)
      COMPLEX*8 A(NP,NP),SUM,DUM2
      INTEGER INDX(N),I,J,IMAX
      REAL*8 D,AAMAX,VV(NMAX),DUM,MAG
C
      D=1.
C
      DO I=1,N
       AAMAX=0.
       DO J=1,N
        IF(MAG(A(I,J)).GT.AAMAX) AAMAX=MAG(A(I,J))
       END DO
       IF(AAMAX.EQ.0.) PAUSE
       VV(I)=1./AAMAX
      END DO
      DO J=1,N
       DO I=1,J-1
        SUM=A(I,J)
        DO K=1,I-1
         SUM=SUM-A(I,K)*A(K,J)
        END DO
        A(I,J)=SUM
       END DO
       AAMAX=0.
       DO I=J,N
        SUM=A(I,J)
        DO K=1,J-1
         SUM=SUM-A(I,K)*A(K,J)
        END DO
        A(I,J)=SUM
        DUM=VV(I)*MAG(SUM)
        IF(DUM.GE.AAMAX) THEN
         IMAX=I
         AAMAX=DUM
        END IF
       END DO
       IF(J.NE.IMAX) THEN
        DO K=1,N
         DUM2=A(IMAX,K)
         A(IMAX,K)=A(J,K)
         A(J,K)=DUM2
        END DO
        D=-D
        VV(IMAX)=VV(J)
       END IF
       INDX(J)=IMAX
       IF(A(J,J).EQ.(0.,0.)) A(J,J)=(TINY,0.)
```

```
        IF(J.NE.N) THEN
         DUM2=1./A(J,J)
         DO I=J+1,N
          A(I,J)=A(I,J)*DUM2
         END DO
        END IF
       END DO
       RETURN
       END  !LUDCMPC
C      ****************************************************************
C      ****************************************************************
       SUBROUTINE LUBKSBC(A,N,NP,INDX,B)
C      ****************************************
C      * COMPLEX VERSION OF LU-BACKSUBSTITUTION   *
C      ****************************************
       INTEGER N,NP
       COMPLEX*8 A(NP,NP),B(N),SUM
       INTEGER INDX(N),II,I,LL
       II=0
       DO I=1,N
        LL=INDX(I)
        SUM=B(LL)
        B(LL)=B(I)
        IF(II.NE.0) THEN
         DO J=II,I-1
          SUM=SUM-A(I,J)*B(J)
         END DO
        ELSE IF(SUM.NE.(0.,0.))  THEN
         II=I
        END IF
        B(I)=SUM
       END DO
       DO I=N,1,-1
        SUM=B(I)
        IF(I.LT.N) THEN
         DO J=I+1,N
          SUM=SUM-A(I,J)*B(J)
         END DO
        END IF
        B(I)=SUM/A(I,I)
       END DO
       RETURN
       END   !LUBKSBC
C      ****************************************************************
C      ****************************************************************
       FUNCTION MAG(A)
C      ***************************************
C      * RETURNS MAGNITUDE OF COMPLEX NUMBER  *
C      ***************************************
       COMPLEX*8 A
       REAL*8 TINY,ZERO,AR,AI,MAG
       PARAMETER( TINY=1.0E-12, ZERO=0.)
       AR=ABS(REAL(A))
       AI=ABS(IMAG(A))
       IF(AR.LT.TINY) THEN AR=ZERO
       IF(AI.LT.TINY) THEN AI=ZERO
       MAG=SQRT(AR**2+AI**2)
       RETURN
```

```
      END   !MAG
C     ***************************************************************
C     ***************************************************************
      SUBROUTINE EIGSRT(N,ARR)
C     ***************************************************************
C     * SORTS COMPLEX ARRAY ARR IN THE ORDER OF DEREASING REAL PARTS *
C     * TERMS WITH EQUAL REAL PARTS ARE SUBSORTED BY IMAGINARY PARTS *
C     ***************************************************************
      PARAMETER (M=7,NSTACK=50,FM=7875.,FA=211.,FC=1663.,FMI=1./FM)
      COMPLEX*8 A,ARR(N)
      INTEGER ISTACK(NSTACK)
      LOGICAL L1,L2,L3
      JSTACK=0
      L=1
      IR=N
      FX=0.
10    IF(IR-L.LT.M)THEN
       DO J=L+1,IR
        A=ARR(J)
        DO I=J-1,1,-1
         L1=REAL(ARR(I)).GT.REAL(A)
         L2=REAL(ARR(I)).EQ.REAL(A)
         L3=IMAG(ARR(I)).GE.IMAG(A)
         IF(L1.OR.(L2.AND.L3))GO TO 12
         ARR(I+1)=ARR(I)
        END DO
        I=0
12      ARR(I+1)=A
       END DO
       IF(JSTACK.EQ.0)RETURN
       IR=ISTACK(JSTACK)
       L=ISTACK(JSTACK-1)
       JSTACK=JSTACK-2
      ELSE
       I=L
       J=IR
       FX=MOD(FX*FA+FC,FM)
       IQ=L+(IR-L+1)*(FX*FMI)
       A=ARR(IQ)
       ARR(IQ)=ARR(L)
20     CONTINUE
21     IF(J.GT.0)THEN
        L1=REAL(A).GT.REAL(ARR(J))
        L2=REAL(A).EQ.REAL(ARR(J))
        L3=IMAG(A).GT.IMAG(ARR(J))
        IF(L1.OR.(L2.AND.L3))THEN
         J=J-1
         GO TO 21
        END IF
       END IF
       IF(J.LE.I)THEN
        ARR(I)=A
        GO TO 30
       END IF
       ARR(I)=ARR(J)
       I=I+1
22     IF(I.LE.N)THEN
        L1=REAL(A).LT.REAL(ARR(I))
```

```
        L2=REAL(A).EQ.REAL(ARR(I))
        L3=IMAG(A).LT.IMAG(ARR(I))
        IF(L1.OR.(L2.AND.L3))THEN
         I=I+1
         GO TO 22
        END IF
       END IF
       IF(J.LE.I)THEN
        ARR(J)=A
        I=J
        GO TO 30
       END IF
       ARR(J)=ARR(I)
       J=J-1
       GO TO 20
30     JSTACK=JSTACK+2
       IF(JSTACK.GT.NSTACK)PAUSE 'NSTACK must be made larger.'
       IF(IR-I.GE.I-L)THEN
        ISTACK(JSTACK)=IR
        ISTACK(JSTACK-1)=I+1
        IR=I-1
       ELSE
        ISTACK(JSTACK)=I-1
        ISTACK(JSTACK-1)=L
        L=I+1
       END IF
      END IF
      GO TO 10
      END !EIGSRT
C     ************************************************************
C     ************************************************************
      SUBROUTINE COPY(A,AEXP,NPA,NAA,B,BEXP,NPB,NAB,NSTATES,NPMAX,NAMAX)
C     ************************************************************************
C     * COPIES A STRING A, AEXP NPA NAA INTO A NEW STRING B BEXP NPB NAB  *
C     ************************************************************************
      INTEGER NPMAX,NAMAX
      INTEGER NPA,NAA,NPB,NAB,NSTATES
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      COMPLEX*8 B(NSTATES,NPMAX+1,NAMAX),BEXP(NAMAX)
      INTEGER I,J,K,L
C
      NPB=NPA
      NAB=NAA
      DO L=1,NAA
       BEXP(L)=AEXP(L)
       DO K=1,NPA+1
        DO I=1,NSTATES
         B(I,K,L)=A(I,K,L)
        END DO
       END DO
      END DO
      RETURN
      END   !COPY
C     ************************************************************************
      SUBROUTINE ADD2(A,AEXP,NPA,NAA,COEFFA,CA,
     *              B,BEXP,NPB,NAB,COEFFB,CB,
     *              NSTATES,NPMAX,NAMAX)
C     ************************************************************
```

```
C     * ADDS VECTOR STRINGS OF EXPONENTIALS A AND B TO PRODUCE     *
C     * THE VECTOR STRING B:=COEFFB*CB*B+COEFFA*CA*A               *
C     *                                                           *
C     *   A---->|COEFFA |----->| CA |--------                     *
C     *          -------        ----          |                   *
C     *         real scalar  real matrix   |---|                  *
C     *                                    | + |-> B:=COEFFB*CB*B *
C     *                                    |---|     +COEFFA*CA*A *
C     *                                      |                    *
C     *   B---->|COEFFB |----->| CB |--------                     *
C     *          -------        ----                             *
C     *         real scalar   real matrix                         *
C     ************************************************************
      INTEGER NPA,NAA,NPB,NAB,NPC
      INTEGER NPMAX,NAMAX,NMAX
      PARAMETER( NMAX=20)
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      COMPLEX*8 B(NSTATES,NPMAX+1,NAMAX),BEXP(NAMAX)
      REAL*8 CA(NSTATES,NSTATES),COEFFA
      REAL*8 CB(NSTATES,NSTATES),COEFFB
      COMPLEX*8 D(NMAX)
      INTEGER I,J,K,L
C
1     FORMAT(' OVERFLOW OF NAMAX IN SUBROUTINE ADD2')
      IF(NAA+NAB.GT.NAMAX) THEN
       WRITE(6,1)
       PAUSE
      END IF
C
C
      NPC=MAX(NPA,NPB)
      DO K=1,NAB
       DO J=1,NPC+1
        IF(J.GT.NPB+1) THEN
         DO I=1,NSTATES
          B(I,J,K)=(0.,0.)
         END DO
        ELSE
           DO I=1,NSTATES
          D(I)=(0.,0.)
          DO L=1,NSTATES
           D(I)=D(I)+COEFFB*CB(I,L)*B(L,J,K)
          END DO
         END DO
         DO I=1,NSTATES
          B(I,J,K)=D(I)
         END DO
        END IF
       END DO
      END DO
C
      DO K=1,NAA
       BEXP(NAB+K)=AEXP(K)
       DO J=1,NPC+1
        IF(J.LE.NPA+1)THEN
         DO I=1,NSTATES
          B(I,J,NAB+K)=(0.,0)
          DO L=1,NSTATES
```

```
            B(I,J,NAB+K)=B(I,J,NAB+K)+COEFFA*CA(I,L)*A(L,J,K)
           END DO
          END DO
         ELSE
          DO I=1,NSTATES
           B(I,J,NAB+K)=(0.,0.)
          END DO
         END IF
        END DO
       END DO
       NPB=NPC
       NAB=NAB+NAA
C
C      SORT INTO DESCENDING VALUES OF REAL PARTS OF EXPONENTS
       CALL QCKSRT(NAB,NPB,BEXP,B,NSTATES,NPMAX,NAMAX)
C
C      COMBINE TERMS WITH CLOSE EXPONENTIAL FACTORS
       CALL SHRINK(B,BEXP,NPB,NAB,0.000001,NSTATES,NPMAX,NAMAX)
C
       RETURN
       END  !ADD2
C      ************************************************************
C      ************************************************************
       SUBROUTINE MULT3(D,A,AEXP,NPA,NAA,B,BEXP,NPB,NAB,C,CEXP,NPC,NAC,
      *                 NSTATES,NPMAX,NAMAX)
C      ****************************************************************
C      * MULTIPLIES CONSTANT VECTOR D WITH STRINGS A AND B TO PRODUCE *
C      * THE STRING C. C=(TRANSP(D)*A)*B                              *
C      * NPA,NPB AND NPC ARE THE DEGREES OF THE POLYNOMIAL            *
C      * COEFFICIENTS OF A,B AND C RESPECTIVELY.                      *
C      * NAA,NAB AND NAC ARE THE NUMBER OF EXPONENTIAL TERMS IN       *
C      * A,B AND C RESPECTIVELY                                       *
C      *                                                              *
C      *    A(t)-->| D |------->  C(t)=((TRANSP(D)*A(t))*B(t)                *
C      *           ---                                                *
C      *            ^                                                 *
C      *    B(t)-----|                                                  *
C      *                                                              *
C      ****************************************************************
       INTEGER NPA,NAA,NPB,NAB,NPC,NAC,NSTATES,NPMAX,NAMAX
       REAL*8 D(NSTATES)
       COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
       COMPLEX*8 B(NSTATES,NPMAX+1,NAMAX),BEXP(NAMAX)
       COMPLEX*8 C(NSTATES,NPMAX+1,NAMAX),CEXP(NAMAX)
       INTEGER I,J,L,M
       INTEGER I1,J1
C
1      FORMAT(' OVERFLOW OF NPMAX IN SUBROUTINE MULT3')
       IF(NPA*NPB.GT.NPMAX) THEN
        WRITE(6,1)
        PAUSE
       END IF
C
C
2      FORMAT(' OVERFLOW OF NAMAX IN SUBROUTINE MULT3')
       IF(NAA+NAB.GT.NAMAX) THEN
        WRITE(6,2)
        PAUSE
```

```
      END IF
C
C
      NPC=NPA+NPB
      NAC=0
      DO I=1,NAA
       DO J=1,NAB
        NAC=NAC+1
        CEXP(NAC)=AEXP(I)+BEXP(J)
        DO L=1,NPA+1
         DO M=1,NPB+1
            DO I1=1,NSTATES
           C(I1,L+M-1,NAC)=(0.,0.)
          END DO
         END DO
        END DO
        DO L=1,NPA+1
         DO M=1,NPB+1
          DO I1=1,NSTATES
           DO J1=1,NSTATES
            C(I1,L+M-1,NAC)=C(I1,L+M-1,NAC)+B(I1,M,J)*D(J1)*A(J1,L,I)
           END DO
          END DO
         END DO
        END DO
       END DO
      END DO
C
C     SORT INTO DESCENDING VALUES OF REAL PARTS OF EXPONENTS
      CALL QCKSRT(NAC,NPC,CEXP,C,NSTATES,NPMAX,NAMAX)
C
C     COMBINE TERMS WITH CLOSE EXPONENTIAL FACTORS
      CALL SHRINK(C,CEXP,NPC,NAC,0.000001,NSTATES,NPMAX,NAMAX)
C
C     ELIMINATE TERMS WITH SMALL COEFFICIENTS
C     CALL TRIM_NA(C,CEXP,NPC,NAC,NSTATES,NPMAX,NAMAX,0.000001)
C     CALL TRIM_NP(C,CEXP,NPC,NAC,NSTATES,NPMAX,NAMAX,0.000001)
      RETURN
      END  !MULT3
C     ************************************************************
C     ************************************************************
      SUBROUTINE ADD(A,AEXP,NPA,NA,B,BEXP,NPB,NB,COEFF,C,NSTATES,
     *                  NPMAX,NAMAX)
C     ************************************************************
C     * ADDS VECTOR STRINGS OF EXPONENTIALS A AND B TO PRODUCE *
C     * THE VECTOR STRING B:=B+COEFF*C*A                       *
C     *                                                        *
C     * B(t)---->|+|------------>  B(t):=B(t)+COEFF*C*A(t)      *
C     *           -                                            *
C     *           ^                                            *
C     *          _|_                                           *
C     *         | C |real matrix                               *
C     *          ---                                           *
C     *           ^                                            *
C     *          __|__                                         *
C     *         |COEFF|real scalar                             *
C     *          -----                                         *
C     *            ^                                           *
```

```
C     * A(t)------|-------------->  A(t)                                  *
C     *                                                                  *
C     ********************************************************************
      INTEGER NPA,NA,NPB,NB,NPC
      INTEGER NPMAX,NAMAX
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      COMPLEX*8 B(NSTATES,NPMAX+1,NAMAX),BEXP(NAMAX)
      REAL*8 C(NSTATES,NSTATES),COEFF
      INTEGER I,J,K,L
C
1     FORMAT(' OVERFLOW OF NAMAX IN SUBROUTINE ADD')
      IF(NA+NB.GT.NAMAX) THEN
       WRITE(6,1)
       PAUSE
      END IF
C
      NPC=MAX(NPA,NPB)
      DO K=1,NB
       DO J=1,NPC+1
        IF(J.GT.NPB+1) THEN
         DO I=1,NSTATES
          B(I,J,K)=(0.,0.)
         END DO
        END IF
       END DO
      END DO
C
      DO K=1,NA
       BEXP(NB+K)=AEXP(K)
       DO J=1,NPC+1
        IF(J.LE.NPA+1)THEN
         DO I=1,NSTATES
          B(I,J,NB+K)=(0.,0)
          DO L=1,NSTATES
           B(I,J,NB+K)=B(I,J,NB+K)+COEFF*C(I,L)*A(L,J,K)
          END DO
         END DO
        ELSE
         DO I=1,NSTATES
          B(I,J,NB+K)=(0.,0.)
         END DO
        END IF
       END DO
      END DO
      NPB=NPC
      NB=NB+NA
C
C     SORT INTO DESCENDING VALUES OF REAL PARTS OF EXPONENTS
C
      CALL QCKSRT(NB,NPB,BEXP,B,NSTATES,NPMAX,NAMAX)
C
C     COMBINE TERMS WITH CLOSE EXPONENTIAL FACTORS
      CALL SHRINK(B,BEXP,NPB,NB,0.000001,NSTATES,NPMAX,NAMAX)
C
      RETURN
      END  !ADD
C     ****************************************************************
C     ****************************************************************
```

```
      SUBROUTINE QCKSRT(N,NP,ARR,BRR,NSTATES,NPMAX,NAMAX)
C     ***************************************************************
C     * SORTS A VECCTOR STRING OF EXPONENTIALS IN THE ORDER OF      *
C     * DEREASING REAL PARTS OF EXPONENTIAL COEFFICIENTS.           *
C     * TERMS WITH EQUAL REAL PARTS ARE SUBSORTED BY IMAGINARY PARTS *
C     ***************************************************************
      PARAMETER (M=7,NSTACK=50,FM=7875.,FA=211.,FC=1663.,FMI=1./FM)
      COMPLEX*8 A,ARR(NAMAX),BRR(NSTATES,NPMAX+1,NAMAX)
      PARAMETER( NMAX=50)
      COMPLEX*8 B(NMAX,NMAX+1)
      INTEGER ISTACK(NSTACK)
      LOGICAL L1,L2,L3
      JSTACK=0
      L=1
      IR=N
      FX=0.
10    IF(IR-L.LT.M)THEN
       DO J=L+1,IR
        A=ARR(J)
        DO K=1,NP+1
         DO M1=1,NSTATES
          B(M1,K)=BRR(M1,K,J)
         END DO
        END DO
        DO I=J-1,1,-1
         L1=REAL(ARR(I)).GT.REAL(A)
         L2=REAL(ARR(I)).EQ.REAL(A)
         L3=IMAG(ARR(I)).GE.IMAG(A)
         IF(L1.OR.(L2.AND.L3))GO TO 12
         ARR(I+1)=ARR(I)
         DO K=1,NP+1
          DO M1=1,NSTATES
           BRR(M1,K,I+1)=BRR(M1,K,I)
          END DO
         END DO
        END DO
        I=0
12      ARR(I+1)=A
        DO K=1,NP+1
         DO M1=1,NSTATES
          BRR(M1,K,I+1)=B(M1,K)
         END DO
        END DO
       END DO
       IF(JSTACK.EQ.0)RETURN
       IR=ISTACK(JSTACK)
       L=ISTACK(JSTACK-1)
       JSTACK=JSTACK-2
      ELSE
       I=L
       J=IR
       FX=MOD(FX*FA+FC,FM)
       IQ=L+(IR-L+1)*(FX*FMI)
       A=ARR(IQ)
       DO K=1,NP+1
        DO M1=1,NSTATES
         B(M1,K)=BRR(M1,K,IQ)
        END DO
```

```
        END DO
        ARR(IQ)=ARR(L)
        DO K=1,NP+1
         DO M1=1,NSTATES
          BRR(M1,K,IQ)=BRR(M1,K,L)
         END DO
        END DO
20      CONTINUE
21      IF(J.GT.0)THEN
         L1=REAL(A).GT.REAL(ARR(J))
         L2=REAL(A).EQ.REAL(ARR(J))
         L3=IMAG(A).GT.IMAG(ARR(J))
         IF(L1.OR.(L2.AND.L3))THEN
          J=J-1
          GO TO 21
         END IF
        END IF
        IF(J.LE.I)THEN
         ARR(I)=A
         DO K=1,NP+1
          DO M1=1,NSTATES
           BRR(M1,K,I)=B(M1,K)
          END DO
         END DO
         GO TO 30
        END IF
        ARR(I)=ARR(J)
        DO K=1,NP+1
         DO M1=1,NSTATES
          BRR(M1,K,I)=BRR(M1,K,J)
         END DO
        END DO
        I=I+1
22      IF(I.LE.N)THEN
         L1=REAL(A).LT.REAL(ARR(I))
         L2=REAL(A).EQ.REAL(ARR(I))
         L3=IMAG(A).LT.IMAG(ARR(I))
         IF(L1.OR.(L2.AND.L3))THEN
          I=I+1
          GO TO 22
         END IF
        END IF
        IF(J.LE.I)THEN
         ARR(J)=A
         DO K=1,NP+1
          DO M1=1,NSTATES
           BRR(M1,K,J)=B(M1,K)
          END DO
         END DO
         I=J
         GO TO 30
        END IF
        ARR(J)=ARR(I)
        DO K=1,NP+1
         DO M1=1,NSTATES
          BRR(M1,K,J)=BRR(M1,K,I)
         END DO
        END DO
```

```
        J=J-1
        GO TO 20
30      JSTACK=JSTACK+2
        IF(JSTACK.GT.NSTACK)PAUSE 'NSTACK must be made larger.'
        IF(IR-I.GE.I-L)THEN
         ISTACK(JSTACK)=IR
         ISTACK(JSTACK-1)=I+1
         IR=I-1
        ELSE
         ISTACK(JSTACK)=I-1
         ISTACK(JSTACK-1)=L
         L=I+1
        END IF
       END IF
       GO TO 10
       END !QCKSRT
C      ***************************************************************
C      ***************************************************************
       FUNCTION DISTANCE(A,B)
C      ***************************************************************
C      * COMPUTES DISTANCE BETWEEN A AND B IN THE COMPLEX PLANE      *
C      ***************************************************************
       COMPLEX*8 A,B
       REAL*8 DISTANCE
       DISTANCE=SQRT((REAL(A)-REAL(B))**2+(IMAG(A)-IMAG(B))**2)
       RETURN
       END  !DISTANCE
C      ***************************************************************
C      ***************************************************************
       SUBROUTINE TRIM_NP(A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX,THRESH)
C      ***************************************************************
C      * REDUCES THE HIGHEST POWER OF t IN THE COEFFICIENT           *
C      * MATRICES IF THE NORMS OF ALL TERMS IN THESE HIGHER          *
C      * POWERS ARE LESS THAN THRESH                                 *
C      * NPA IS MODIFIED                                             *
C      ***************************************************************
       PARAMETER( E=2.71828182845904523536)
       INTEGER NPMAX,NAMAX
       INTEGER NPA,NAA,I,J,K,JMAX
       COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
       REAL*8 AR,AI,SIGMA,ATEMP
       IF(NPA.EQ.1)THEN
        RETURN
       END IF
       JMAX=NPA+1
       DO J=JMAX,2,-1
        DO K=1,NAA
         SIGMA=REAL(AEXP(K))
         IF(SIGMA.NE.0.)THEN
          DO I=1,NSTATES
           AR=REAL(A(I,J,K))
           AI=IMAG(A(I,J,K))
           ATEMP=SQRT(AR*AR+AI*AI)
           ATEMP=ATEMP*(-(J-1)/(E*SIGMA))**(J-1)
           IF(ATEMP.GT.THRESH) THEN
            NPA=J-1
            RETURN
           END IF
```

```
          END DO
         END IF
        END DO
       END DO
       RETURN
       END  !TRIM_NP
C      ************************************************************
C      ************************************************************
       SUBROUTINE TRIM_NA(A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX,THRESH)
C      ************************************************************
C      * ELIMINATES ANY EXPONENTIAL TERM IN A IF THE NORMS        *
C      * OF ALL THE COEFFICINT ELEMENTS ARE LESS THAN THRESH.     *
C      * A,AEXP AND NA ARE MODIFIED                               *
C      ************************************************************
       PARAMETER( E=2.71828182845904523536)
       INTEGER NPMAX,NAMAX
       INTEGER NPA,NAA,I,J,K,M
       COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
       REAL*8 AR,AI,SIGMA,ATEMP,AN
       M=0
       DO K=1,NAA
C
        SIGMA=REAL(AEXP(K))
         AN=0.
         DO J=1,NPA+1
          DO I=1,NSTATES
           AR=REAL(A(I,J,K))
           AI=IMAG(A(I,J,K))
           ATEMP=SQRT(AR*AR+AI*AI)
           IF((SIGMA.LT.0.).AND.(J.GT.1)) THEN
             ATEMP=ATEMP*(-(J-1)/(E*SIGMA))**(J-1)
           END IF
           IF(ATEMP.GT.AN) THEN
            AN=ATEMP
           END IF
          END DO
         END DO
C
        IF(AN.LT.THRESH) THEN
         M=M+1
        ELSE
         AEXP(K-M)=AEXP(K)
         DO J=1,NPA+1
          DO I=1,NSTATES
           A(I,J,K-M)=A(I,J,K)
          END DO
         END DO
        END IF
       END DO
       NAA=NAA-M
       RETURN
       END   !TRIM_NA
C      ************************************************************
C      ************************************************************
       SUBROUTINE SHRINK(A,AEXP,NPA,NAA,EPSILON,NSTATES,NPMAX,NAMAX)
C      ************************************************************
C      * COMBINES TERMS IN A VECTOR STRING IF THE EXPONENT COEFFICIENTS *
C      * ARE WITHIN EPSILON OF EACH OTHER.                              *
```

```
C     ****************************************************************
      INTEGER I,J,K,L,NAA,NPA,NSTATES,NPMAX,NAMAX
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      REAL*8 EPSILON
      J=1
      DO I=1,NAA-1
       IF(DISTANCE(AEXP(I),AEXP(I+1)).GE.EPSILON) THEN
        J=J+1
        AEXP(J)=AEXP(I+1)
        DO K=1,NPA+1
         DO L=1,NSTATES
          A(L,K,J)=A(L,K,I+1)
         END DO
        END DO
       ELSE
        DO K=1,NPA+1
         DO L=1,NSTATES
          A(L,K,J)=A(L,K,J)+A(L,K,I+1)
         END DO
        END DO
       END IF
      END DO
      NAA=J
      RETURN
      END  !SHRINK
C     ****************************************************************
C     ****************************************************************
      SUBROUTINE MULT(A,AEXP,NPA,NAA,B,BEXP,NPB,NAB,TEMP,TEMPEXP,
     *                NSTATES,NPMAX,NAMAX,LT,LN)
C     ****************************************************************
C     * MULTIPLIES STRINGS A AND B AND PLACES IT IN B                *
C     * NPA AND NPB ARE THE DEGREES OF THE POLYNOMIAL                *
C     * COEFFICIENTS OF A AND B RESPECTIVELY.                        *
C     * NAA AND NAB ARE THE NUMBER OF EXPONENTIAL TERMS IN           *
C     * A AND B RESPECTIVELY                                         *
C     * IF LT=1 THEN A-TRANSPOSED IS THE COEFFICIENT                 *
C     * IF LT=0 THEN A IS THE COEFFICIENT                            *
C     * IF LN=1 THEN A(-t) IS USED INSTEAD OF A(t)                   *
C     * IF LN=0 THEN A(t) IS USED                                    *
C     *                                                              *
C     *                                                              *
C     *                      _______                                 *
C     *                     |       |                                *
C     *      B(t)---------->| A(t)  |--------> B(t):=A(t)*B(t)       *
C     *   vector string     |-------|            vector string       *
C     *                  matrix string                               *
C     *                                                              *
C     ****************************************************************
      INTEGER NSTATES,NPA,NAA,NPB,NAB,NPC,NAC,NPMAX,NAMAX
      COMPLEX*8 A(NSTATES,NSTATES,NSTATES+1,NSTATES),AEXP(NSTATES)
      COMPLEX*8 B(NSTATES,NPMAX+1,NAMAX),BEXP(NAMAX)
      COMPLEX*8 TEMP(NSTATES,NPMAX+1,NAMAX),TEMPEXP(NAMAX)
      INTEGER I,J,K,L,M
      INTEGER I1,J1
      INTEGER LT,LN
C
C
1     FORMAT(' OVERFLOW OF NAMAX IN SUBROUTINE MULT')
      IF(NAA*NAB.GT.NAMAX) THEN
```

```
        WRITE(6,1)
        PAUSE
       END IF
C
C
2      FORMAT(' OVERFLOW OF NPMAX IN SUBROUTINE MULT')
       IF(NPA+NPB.GT.NPMAX) THEN
        WRITE(6,2)
        PAUSE
       END IF
C
       DO K=1,NAB
        TEMPEXP(K)=BEXP(K)
        DO J=1,NPB+1
         DO I=1,NSTATES
          TEMP(I,J,K)=B(I,J,K)
         END DO
        END DO
       END DO
C
       NPC=NPA+NPB
       NAC=0
       DO I=1,NAA
        DO J=1,NAB
         NAC=NAC+1
         BEXP(NAC)=(1-2*LN)*AEXP(I)+TEMPEXP(J)
         DO L=1,NPA+1
          DO M=1,NPB+1
             DO I1=1,NSTATES
            B(I1,L+M-1,NAC)=(0.,0.)
           END DO
          END DO
         END DO
         DO L=1,NPA+1
          DO M=1,NPB+1
           DO I1=1,NSTATES
            DO J1=1,NSTATES
             IF(LT.EQ.0) THEN
              B(I1,L+M-1,NAC)=B(I1,L+M-1,NAC)
     *                 +((1-2*LN)**(L+1))*A(I1,J1,L,I)*TEMP(J1,M,J)
             ELSE
              B(I1,L+M-1,NAC)=B(I1,L+M-1,NAC)
     *                 +((1-2*LN)**(L+1))*A(J1,I1,L,I)*TEMP(J1,M,J)
             END IF
            END DO
           END DO
          END DO
         END DO
        END DO
       END DO
       NPB=NPC
       NAB=NAC
C
C
C      SORT INTO DESCENDING VALUES OF REAL PARTS OF EXPONENTS
C
       CALL QCKSRT(NAB,NPB,BEXP,B,NSTATES,NPMAX,NAMAX)
C
```

```
C     COMBINE TERMS WITH CLOSE EXPONENTIAL FACTORS
      CALL SHRINK(B,BEXP,NPB,NAB,0.000001,NSTATES,NPMAX,NAMAX)
C
C     REMOVE SMALL TERMS
      CALL TRIM_NA(B,BEXP,NPB,NAB,NSTATES,NPMAX,NAMAX,0.00001)
C
      CALL TRIM_NP(B,BEXP,NPB,NAB,NSTATES,NPMAX,NAMAX,0.00001)
C
      RETURN
      END  !MULT
C     ****************************************************************
C     ****************************************************************
      SUBROUTINE INT_T_INF(A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX)
C     ****************************************************************
C     * INTEGRATES A VECTOR STRING OF EXPONENTIALS FROM t            *
C     * TO INFINITY.                                                 *
C     * NSTATES=NUMBER OF STATE COMPONENTS.                          *
C     * NPA=HIGHEST POWER OF t IN A.                                 *
C     * NAA=NUMBER OF EXPONENTIAL TERMS IN A.                        *
C     * STABILITY OF FTILDA MATRIX SHOULD INSURE THAT THE            *
C     * INTEGRALS EXIST.                                             *
C     *                                                              *
C     *                                                              *
C     *                                                              *
C     *      A(t)->| INTEGRAL FROM t TO INFINITY OF A(s) ds |->A(t)  *
C     *             ----------------------------------------         *
C     *                                                              *
C     *                                                              *
C     *                                                              *
C     ****************************************************************
      INTEGER NPA,NAA,NSTATES,NPMAX,NAMAX
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      INTEGER I,J,K
1     FORMAT('  INTEGRAL DOES NOT EXIST')
      DO K=1,NAA
       ZERO=0.
       IF(REAL(AEXP(K)).GE.ZERO) THEN
        WRITE(6,1)
        PAUSE
       END IF
      END DO
      DO K=1,NAA
       DO I=1,NSTATES
        A(I,NPA+1,K)=-A(I,NPA+1,K)/AEXP(K)
       END DO
       DO J=NPA,1,-1
        DO I=1,NSTATES
         A(I,J,K)=-(A(I,J,K)+J*A(I,J+1,K))/AEXP(K)
        END DO
       END DO
      END DO
      CALL TRIM_NP(A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX,0.00001)
      RETURN
      END  !INT_T_INF
C     ****************************************************************
C     ****************************************************************
      SUBROUTINE INT_0_T(X,A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX)
C     ****************************************************************
```

```
C     * INTEGRATES A VECTOR STRING FROM 0 TO t AND ADDS THE                *
C     * CONSTANT VECTOR X.                                                 *
C     * NSTATES=NUMBER OF STATE COMPONENTS.                                *
C     * NPA=HIGHEST POWER OF t IN A.                                       *
C     * NAA=NUMBER OF EXPONENTIAL TERMS IN A.                              *
C     *                                                                    *
C     **********************************************************************
      INTEGER NPA,NAA,NSTATES,NPMAX,NAMAX,NSTATEMAX
      PARAMETER ( NSTATEMAX=6)
      REAL*8 X(NSTATES)
      COMPLEX*8 A(NSTATES,NPMAX+1,NAMAX),AEXP(NAMAX)
      COMPLEX*8 C(NSTATEMAX)
      COMPLEX*8 ZERO
      INTEGER I,J,K
C
1     FORMAT(' MUST INCREASE NSTATEMAX IN INT_0_T ROUTINE')
      IF(NSTATES.GT.NSTATEMAX) THEN
       WRITE(6,1)
       PAUSE
      END IF
C
C
2     FORMAT(' OVERFLOW OF NPMAX IN SUBROUTINE INT_0_T')
C
      ZERO=(0.,0.)
C
C     INITIALIZE INTEGRAL AT STATE DEVIATION FROM TRIM
      DO I=1,NSTATES
       C(I)=X(I)
      END DO
C
      DO K=1,NAA
       IF(AEXP(K).NE.ZERO) THEN
        DO I=1,NSTATES
         A(I,NPA+1,K)=A(I,NPA+1,K)/AEXP(K)
        END DO
        DO J=NPA,1,-1
         DO I=1,NSTATES
          A(I,J,K)=(A(I,J,K)-J*A(I,J+1,K))/AEXP(K)
         END DO
        END DO
       ELSE
        IF(NPA+1.GT.NPMAX) THEN
         WRITE(6,2)
         PAUSE
        END IF
        NPA=NPA+1
        DO I=1,NSTATES
         DO J=NPA+1,2,-1
          A(I,J,K)=A(I,J-1,K)/(J-1)
         END DO
         A(I,1,K)=ZERO
        END DO
       END IF
       DO I=1,NSTATES
        C(I)=C(I)-A(I,1,K)
       END DO
      END DO
```

```
C
C     APPEND e^0*t TERM TO SATISFY t=0 CONDITION ON INTEGRAL
      NAA=NAA+1
      AEXP(NAA)=ZERO
      DO I=1,NSTATES
       A(I,1,NAA)=C(I)
       DO J=2,NPA+1
        A(I,J,NAA)=ZERO
       END DO
      END DO
      CALL TRIM_NP(A,AEXP,NPA,NAA,NSTATES,NPMAX,NAMAX,0.00001)
C
      RETURN
      END  !INT_0_T
C     ************************************************
      SUBROUTINE ZEROEVAL(S,NPS,NAS,ZE,NSTATES,NPMAX,NAMAX)
C     ************************************************
C     * EVALUATES A VECTOR STRING OF EXPONENTIALS AT t=0 *
C     ************************************************
      INTEGER NSTATES,NPS,NAS,MPMAX,NAMAX,I,J
      COMPLEX*8 S(NSTATES,NPMAX+1,NAMAX)
      REAL*8 ZE(NSTATES)
      DO I=1,NSTATES
       ZE(I)=0.
       DO J=1,NAS
        ZE(I)=ZE(I)+REAL(S(I,1,J))
       END DO
      END DO
      RETURN
      END !ZEROEVAL
C     ******************************************************
      SUBROUTINE TESTP(A,Q,R,P,N)
C     ******************************************************
C     * TESTS SOLUTION FOR RICCATI EQUATION OBTAINED FROM  *
C     * P=Z2*Z1INVERSE BY INSERTING THE SOLUTION BACK INTO *
C     * THE EQUATION A'*P+P*A+Q+P*R*P=0 AND WRITING OUT    *
C     * ERROR                                              *
C     ******************************************************
      REAL*8 A(N,N),Q(N,N),R(N,N),P(N,N),E(100,100)
      INTEGER I,J,K,L
1     FORMAT(' RICCATI ERROR(',I2,',',I2,')=',E25.17)
      DO I=1,N
       DO J=1,N
        E(I,J)=Q(I,J)
        DO K=1,N
         E(I,J)=E(I,J)+A(K,I)*P(K,J)+A(K,J)*P(K,I)
         DO L=1,N
          E(I,J)=E(I,J)+P(K,I)*R(K,L)*P(L,J)
         END DO
        END DO
        WRITE(6,1) I,J,E(I,J)
       END DO
      END DO
      RETURN
      END  ! TESTP
C     ************************************************
      SUBROUTINE TESTF(A,R,P,F,N)
C     ************************************************
```

```
C     * TESTS THE SOLUTION FOR FTILDA OBTAINED FROM   *
C     * Z1*EIG*Z1INVERSE BY CALCULATION OF THE        *
C     * EQUIVALENT FORM ATILDA+RTILDA*P               *
C     * AND WRITES OUT THE ERROR                      *
C     ************************************************
      REAL*8 A(N,N),R(N,N),P(N,N),F(N,N),E(100,100)
      INTEGER I,J,K
1     FORMAT(' FTILDA ERROR(',I2,',',I2,')=',E25.17)
      DO I=1,N
       DO J=1,N
        E(I,J)=A(I,J)-F(I,J)
        DO K=1,N
         E(I,J)=E(I,J)+R(I,K)*P(K,J)
        END DO
        WRITE(6,1) I,J,E(I,J)
       END DO
      END DO
      RETURN
      END  ! TESTF
C     ************************************************
      SUBROUTINE MMULT(C,A,AEXP,NP,NA,NPMAX,NAMAX,N)
C     MULTIPLIES STRING A TIME CONCTANT REAL MATRIX C
      INTEGER I,K,L,M,N,NPMAX,MAMAX
      REAL*8 C(N,N)
      COMPLEX*8  A(N,NPMAX+1,NAMAX),AEXP(NAMAX),TEMP(100)
      DO L=1,NA
       DO K=1,NP+1
        DO I=1,N
         TEMP(I)=0.
         DO M=1,N
          TEMP(I)=TEMP(I)+C(I,M)*A(M,K,L)
         END DO
        END DO
        DO I=1,N
         A(I,K,L)=TEMP(I)
        END DO
       END DO
      END DO
      RETURN
      END !MMULT
C*********************************************************************
C*********************************************************************
      SUBROUTINE WRITEXP(A,AEXP,NP,NA,N)
C     *****************************************
C     * WRITES OUT STRINGS OF MATRIX EXPONENTIALS *
C     *****************************************
      INTEGER NA,NP,N
      INTEGER I,J,K,L
      COMPLEX*8 A(N,N,N+1,N),AEXP(N)
1     FORMAT(' EXPONENT IS',E25.17,',',E25.17)
2     FORMAT(' POWER OF t IS ',I2)
3     FORMAT(' MATRIX IS ')
4     FORMAT(E25.17,',',E25.17)
5     FORMAT(' ')
6     FORMAT(' NA=',I2)
7     FORMAT(' NP=',I2)
8     FORMAT(' N=',I2)
C
```

```
C     OUTPUT MATRIX EXPONENTIAL
      WRITE(6,6) NA
      WRITE(6,7) NP
      WRITE(6,8) N
      DO L=1,NA
       WRITE(6,1) REAL(AEXP(L)),IMAG(AEXP(L))
       DO K=1,NP+1
        WRITE(6,2) K-1
        WRITE(6,3)
        DO I=1,N
         WRITE(6,4) (REAL(A(I,J,K,L)),IMAG(A(I,J,K,L)),J=1,N)
         WRITE(6,5)
        END DO
       END DO
      END DO
C
      RETURN
      END !WRITEXP
C     ***************************************************************
C     ***************************************************************
      SUBROUTINE WRITE_ZQ(Z,ZEXP,NPZ,NAZ,Q,QEXP,NPQ,NAQ,NSTATES,
     *                    NPMAX,NAMAX)
      INTEGER NPZ,NAZ,NPQ,NAQ,NSTATES,NPMAX,NAMAX,I,K,L
      COMPLEX*8 Z(NSTATES,NPMAX+1,NAMAX),ZEXP(NAMAX)
      COMPLEX*8 Q(NSTATES,NPMAX+1,NAMAX),QEXP(NAMAX)
C
1     FORMAT('     ')
2     FORMAT('     NAZ=',I6,'       NPZ=',I3)
3     FORMAT('     NAQ=',I6,'       NPQ=',I3)
4     FORMAT(' Z(',I2,',',I2,',',I6,')=',E25.17,' , ',E25.17)
5     FORMAT(' Q(',I2,',',I2,',',I6,')=',E25.17,' , ',E25.17)
6     FORMAT(' EXPONENT=',E25.17,',',E25.17)
       WRITE(6,1)
       WRITE(6,2) NAZ,NPZ
       WRITE(6,3) NAQ,NPQ
       WRITE(6,1)
       DO L=1,NAZ
        WRITE(6,1)
        WRITE(6,6) REAL(ZEXP(L)),IMAG(ZEXP(L))
        DO K=1,NPZ+1
         DO I=1,NSTATES
          WRITE(6,4) I,K,L,REAL(Z(I,K,L)),IMAG(Z(I,K,L))
         END DO
        END DO
       END DO
       WRITE(6,1)
       DO L=1,NAQ
        WRITE(6,1)
        WRITE(6,6) REAL(QEXP(L)),IMAG(QEXP(L))
        DO K=1,NPQ+1
         DO I=1,NSTATES
          WRITE(6,5) I,K,L,REAL(Q(I,K,L)),IMAG(Q(I,K,L))
         END DO
        END DO
       END DO
      RETURN
      END  !WRITE_ZQ
C     ***************************************************************
```

```
C     ************************************************************
C     ***********************************************
C*****************************************************************
C           ZZ_RG     - The eispack eigenvalue/eigenvector routines.
C           ZZ_ELMHES
C           ZZ_HQR
C           ZZ_ELTRAN
C           ZZ_HQR2
C           ZZ_BALBAK
C
C      ************************************************
C
C
C     THIS ROUTINE CONTAINS GENERALIZED EIGENVECTOR MODIFICATIONS MADE
C     B. MEARS  11/14/82
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_RG(NM,N,A,WR,WI,MATZ,Z,IV1,FV1,IERR)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER N,NM,IS1,IS2,IERR,MATZ
      DIMENSION A(NM,N),WR(N),WI(N),Z(NM,N),FV1(N)
      INTEGER IV1(N)
C
C     THIS SUBROUTINE CALLS THE RECOMMENDED SEQUENCE OF
C     SUBROUTINES FROM THE EIGENSYSTEM SUBROUTINE PACKAGE (EISPACK)
C     TO FIND THE EIGENVALUES AND EIGENVECTORS (IF DESIRED)
C     OF A REAL GENERAL MATRIX.
C
C     ON INPUT-
C
C        NM  MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C        ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C        DIMENSION STATEMENT,
C
C        N  IS THE ORDER OF THE MATRIX  A,
C
C        A  CONTAINS THE REAL GENERAL MATRIX,
C
C        MATZ  IS AN INTEGER VARIABLE SET EQUAL TO ZERO IF
C        ONLY EIGENVALUES ARE DESIRED,  OTHERWISE IT IS SET TO
C        ANY NON-ZERO INTEGER FOR BOTH EIGENVALUES AND EIGENVECTORS.
C
C     ON OUTPUT-
C
C        WR  AND  WI  CONTAIN THE REAL AND IMAGINARY PARTS,
C        RESPECTIVELY, OF THE EIGENVALUES.  COMPLEX CONJUGATE
C        PAIRS OF EIGENVALUES APPEAR CONSECUTIVELY WITH THE
C        EIGENVALUE HAVING THE POSITIVE IMAGINARY PART FIRST,
C
C        Z  CONTAINS THE REAL AND IMAGINARY PARTS OF THE EIGENVECTORS
C        IF MATZ IS NOT ZERO.  IF THE J-TH EIGENVALUE IS REAL, THE
C        J-TH COLUMN OF  Z  CONTAINS ITS EIGENVECTOR.  IF THE J-TH
C        EIGENVALUE IS COMPLEX WITH POSITIVE IMAGINARY PART, THE
C        J-TH AND (J+1)-TH COLUMNS OF  Z  CONTAIN THE REAL AND
C        IMAGINARY PARTS OF ITS EIGENVECTOR.  THE CONJUGATE OF THIS
C        VECTOR IS THE EIGENVECTOR FOR THE CONJUGATE EIGENVALUE,
```

```
C
C        IERR  IS AN INTEGER OUTPUT VARIABLE SET EQUAL TO AN
C        ERROR COMPLETION CODE DESCRIBED IN SECTION 2B OF THE
C        DOCUMENTATION.  THE NORMAL COMPLETION CODE IS ZERO,
C
C        IV1  AND  FV1  ARE TEMPORARY STORAGE ARRAYS.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
      IF (N .LE. NM) GO TO 10
      IERR = 10 * N
      GO TO 50
C
   10 CALL  ZZ_BALANC(NM,N,A,IS1,IS2,FV1)
      CALL  ZZ_ELMHES(NM,N,IS1,IS2,A,IV1)
      IF (MATZ .NE. 0) GO TO 20
C     ********** FIND EIGENVALUES ONLY **********
      CALL  ZZ_HQR(NM,N,IS1,IS2,A,WR,WI,IERR)
      GO TO 50
C     ********** FIND BOTH EIGENVALUES AND EIGENVECTORS **********
   20 CALL  ZZ_ELTRAN(NM,N,IS1,IS2,A,IV1,Z)
      CALL  ZZ_HQR2(NM,N,IS1,IS2,A,WR,WI,Z,IERR)
      IF (IERR .NE. 0) GO TO 50
      CALL  ZZ_BALBAK(NM,N,IS1,IS2,FV1,N,Z)
   50 RETURN
C     ********** LAST CARD OF ZZ_RG **********
      END
*DECK ZZ_BALANC
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_BALANC(NM,N,A,LOW,IGH,SCALE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER I,J,K,L,M,N,JJ,NM,IGH,LOW,IEXC
      DIMENSION A(NM,N),SCALE(N)
      LOGICAL NOCONV
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ZZ_BALANCE,
C     NUM. MATH. 13, 293-304(1969) BY PARLETT AND REINSCH.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 315-326(1971).
C
C     THIS SUBROUTINE ZZ_BALANCES A REAL MATRIX AND ISOLATES
C     EIGENVALUES WHENEVER POSSIBLE.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
C
C        A CONTAINS THE INPUT MATRIX TO BE ZZ_BALANCED.
C
```

```
C     ON OUTPUT-
C
C        A CONTAINS THE ZZ_BALANCED MATRIX,
C
C        LOW AND IGH ARE TWO INTEGERS SUCH THAT A(I,J)
C          IS EQUAL TO ZERO IF
C           (1) I IS GREATER THAN J AND
C           (2) J=1,...,LOW-1 OR I=IGH+1,...,N,
C
C        SCALE CONTAINS INFORMATION DETERMINING THE
C           PERMUTATIONS AND SCALING FACTORS USED.
C
C     SUPPOSE THAT THE PRINCIPAL SUBMATRIX IN ROWS LOW THROUGH IGH
C     HAS BEEN ZZ_BALANCED, THAT P(J) DENOTES THE INDEX INTERCHANGED
C     WITH J DURING THE PERMUTATION STEP, AND THAT THE ELEMENTS
C     OF THE DIAGONAL MATRIX USED ARE DENOTED BY D(I,J).  THEN
C        SCALE(J) = P(J),    FOR J = 1,...,LOW-1
C                 = D(J,J),      J = LOW,...,IGH
C                 = P(J)         J = IGH+1,...,N.
C     THE ORDER IN WHICH THE INTERCHANGES ARE MADE IS N TO IGH+1,
C     THEN 1 TO LOW-1.
C
C     NOTE THAT 1 IS RETURNED FOR IGH IF IGH IS ZERO FORMALLY.
C
C     THE ALGOL PROCEDURE EXC CONTAINED IN ZZ_BALANCE APPEARS IN
C     ZZ_BALANC  IN LINE.  (NOTE THAT THE ALGOL ROLES OF IDENTIFIERS
C     K,L HAVE BEEN REVERSED.)
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
C     ********** RADIX IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                THE BASE OF THE MACHINE FLOATING POINT REPRESENTATION.
C
C                **********
      RADIX = 2.
C
      B2 = RADIX * RADIX
      K = 1
      L = N
      GO TO 100
C     ********** IN-LINE PROCEDURE FOR ROW AND
C                COLUMN EXCHANGE **********
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
      DO 30 I = 1, L
         F = A(I,J)
         A(I,J) = A(I,M)
         A(I,M) = F
   30 CONTINUE
C
      DO 40 I = K, N
         F = A(J,I)
         A(J,I) = A(M,I)
         A(M,I) = F
```

```
   40 CONTINUE
C
   50 GO TO (80,130), IEXC
C     ********** SEARCH FOR ROWS ISOLATING AN EIGENVALUE
C                AND PUSH THEM DOWN **********
   80 IF (L .EQ. 1) GO TO 280
      L = L - 1
C     ********** FOR J=L STEP -1 UNTIL 1 DO -- **********
  100 DO 120 JJ = 1, L
         J = L + 1 - JJ
C
         DO 110 I = 1, L
            IF (I .EQ. J) GO TO 110
            IF (A(J,I) .NE. 0.0) GO TO 120
  110    CONTINUE
C
         M = L
         IEXC = 1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ********** SEARCH FOR COLUMNS ISOLATING AN EIGENVALUE
C                AND PUSH THEM LEFT **********
  130 K = K + 1
C
  140 DO 170 J = K, L
C
         DO 150 I = K, L
            IF (I .EQ. J) GO TO 150
            IF (A(I,J) .NE. 0.0) GO TO 170
  150    CONTINUE
C
         M = K
         IEXC = 2
         GO TO 20
  170 CONTINUE
C     ********** NOW ZZ_BALANCE THE SUBMATRIX IN ROWS K TO L **********
      DO 180 I = K, L
  180 SCALE(I) = 1.0
C     ********** ITERATIVE LOOP FOR NORM REDUCTION **********
  190 NOCONV = .FALSE.
C
      DO 270 I = K, L
         C = 0.0
         R = 0.0
C
         DO 200 J = K, L
            IF (J .EQ. I) GO TO 200
            C = C + ABS(A(J,I))
            R = R + ABS(A(I,J))
  200    CONTINUE
C     ********** GUARD AGAINST ZERO C OR R DUE TO UNDERFLOW **********
         IF (C .EQ. 0.0 .OR. R .EQ. 0.0) GO TO 270
         G = R / RADIX
         F = 1.0
         S = C + R
  210    IF (C .GE. G) GO TO 220
```

```
          F = F * RADIX
          C = C * B2
          GO TO 210
  220     G = R * RADIX
  230     IF (C .LT. G) GO TO 240
          F = F / RADIX
          C = C / B2
          GO TO 230
C      ********** NOW ZZ_BALANCE **********
  240     IF ((C + R) / F .GE. 0.95 * S) GO TO 270
          G = 1.0 / F
          SCALE(I) = SCALE(I) * F
          NOCONV = .TRUE.
C
          DO 250 J = K, N
  250     A(I,J) = A(I,J) * G
C
          DO 260 J = 1, L
  260     A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
C     ********** LAST CARD OF ZZ_BALANC **********
      END
*DECK ZZ_ELMHES
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_ELMHES(NM,N,LOW,IGH,A,INT)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NM,N),INT(IGH)
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ZZ_ELMHES,
C     NUM. MATH. 12, 349-368(1968) BY MARTIN AND WILKINSON.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 339-358(1971).
C
C     GIVEN A REAL GENERAL MATRIX, THIS SUBROUTINE
C     REDUCES A SUBMATRIX SITUATED IN ROWS AND COLUMNS
C     LOW THROUGH IGH TO UPPER HESSENBEZZ_RG FORM BY
C     STABILIZED ELEMENTARY SIMILARITY TRANSFORMATIONS.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
C
C        LOW AND IGH ARE INTEGERS DETERMINED BY THE ZZ_BALANCING
C          SUBROUTINE  ZZ_BALANC.  IF  ZZ_BALANC  HAS NOT BEEN USED,
C          SET LOW=1, IGH=N,
```

```
C
C        A CONTAINS THE INPUT MATRIX.
C
C     ON OUTPUT-
C
C        A CONTAINS THE HESSENBEZZ_RG MATRIX.  THE MULTIPLIERS
C          WHICH WERE USED IN THE REDUCTION ARE STORED IN THE
C          REMAINING TRIANGLE UNDER THE HESSENBEZZ_RG MATRIX,
C
C        INT CONTAINS INFORMATION ON THE ROWS AND COLUMNS
C          INTERCHANGED IN THE REDUCTION.
C          ONLY ELEMENTS LOW THROUGH IGH ARE USED.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
      LA = IGH - 1
      KP1 = LOW + 1
      IF (LA .LT. KP1) GO TO 200
C
      DO 180 M = KP1, LA
         MM1 = M - 1
         X = 0.0
         I = M
C
         DO 100 J = M, IGH
            IF (ABS(A(J,MM1)) .LE. ABS(X)) GO TO 100
            X = A(J,MM1)
            I = J
  100    CONTINUE
C
         INT(M) = I
         IF (I .EQ. M) GO TO 130
C    ********** INTERCHANGE ROWS AND COLUMNS OF A **********
         DO 110 J = MM1, N
            Y = A(I,J)
            A(I,J) = A(M,J)
            A(M,J) = Y
  110    CONTINUE
C
         DO 120 J = 1, IGH
            Y = A(J,I)
            A(J,I) = A(J,M)
            A(J,M) = Y
  120    CONTINUE
C    ********** END INTERCHANGE **********
  130    IF (X .EQ. 0.0) GO TO 180
         MP1 = M + 1
C
         DO 160 I = MP1, IGH
            Y = A(I,MM1)
            IF (Y .EQ. 0.0) GO TO 160
            Y = Y / X
            A(I,MM1) = Y
C
            DO 140 J = M, N
```

```
  140       A(I,J) = A(I,J) - Y * A(M,J)
C
            DO 150 J = 1, IGH
  150       A(J,M) = A(J,M) + Y * A(J,I)
C
  160    CONTINUE
C
  180 CONTINUE
C
  200 RETURN
C    ********** LAST CARD OF ZZ_ELMHES **********
      END
*DECK ZZ_HQR
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_HQR(NM,N,LOW,IGH,H,WR,WI,IERR)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER EN,ENM2
      DIMENSION H(NM,N),WR(N),WI(N)
      DOUBLE PRECISION NORM,MACHEP
      LOGICAL NOTLAS
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ZZ_HQR,
C     NUM. MATH. 14, 219-231(1970) BY MARTIN, PETERS, AND WILKINSON.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 359-371(1971).
C
C     THIS SUBROUTINE FINDS THE EIGENVALUES OF A REAL
C     UPPER HESSENBEZZ_RG MATRIX BY THE QR METHOD.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
C
C        LOW AND IGH ARE INTEGERS DETERMINED BY THE ZZ_BALANCING
C          SUBROUTINE  ZZ_BALANC.  IF  ZZ_BALANC  HAS NOT BEEN USED,
C          SET LOW=1, IGH=N,
C
C        H CONTAINS THE UPPER HESSENBEZZ_RG MATRIX.  INFORMATION ABOUT
C          THE TRANSFORMATIONS USED IN THE REDUCTION TO HESSENBEZZ_RG
C          FORM BY  ZZ_ELMHES  OR  ORTHES, IF PERFORMED, IS STORED
C          IN THE REMAINING TRIANGLE UNDER THE HESSENBEZZ_RG MATRIX.
C
C     ON OUTPUT-
C
C        H HAS BEEN DESTROYED.  THEREFORE, IT MUST BE SAVED
C          BEFORE CALLING  ZZ_HQR  IF SUBSEQUENT CALCULATION AND
C          BACK TRANSFORMATION OF EIGENVECTORS IS TO BE PERFORMED,
C
C        WR AND WI CONTAIN THE REAL AND IMAGINARY PARTS,
C          RESPECTIVELY, OF THE EIGENVALUES.  THE EIGENVALUES
C          ARE UNORDERED EXCEPT THAT COMPLEX CONJUGATE PAIRS
C          OF VALUES APPEAR CONSECUTIVELY WITH THE EIGENVALUE
```

```
C          HAVING THE POSITIVE IMAGINARY PART FIRST.  IF AN
C          ERROR EXIT IS MADE, THE EIGENVALUES SHOULD BE CORRECT
C          FOR INDICES IERR+1,...,N,
C
C        IERR IS SET TO
C          ZERO       FOR NORMAL RETURN,
C          J          IF THE J-TH EIGENVALUE HAS NOT BEEN
C                     DETERMINED AFTER 30 ITERATIONS.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
C     ********** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
C
C                **********
      MACHEP = 2.**(-47)
C
      IERR = 0
      NORM = 0.0
      K = 1
C     ********** STORE ROOTS ISOLATED BY ZZ_BALANC
C                AND COMPUTE MATRIX NORM **********
      DO 50 I = 1, N
C
         DO 40 J = K, N
   40    NORM = NORM + ABS(H(I,J))
C
         K = I
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 50
         WR(I) = H(I,I)
         WI(I) = 0.0
   50 CONTINUE
C
      EN = IGH
      T = 0.0
C     ********** SEARCH FOR NEXT EIGENVALUES **********
   60 IF (EN .LT. LOW) GO TO 1001
      ITS = 0
      NA = EN - 1
      ENM2 = NA - 1
C     ********** LOOK FOR SINGLE SMALL SUB-DIAGONAL ELEMENT
C                FOR L=EN STEP -1 UNTIL LOW DO -- **********
   70 DO 80 LL = LOW, EN
         L = EN + LOW - LL
         IF (L .EQ. LOW) GO TO 100
         S = ABS(H(L-1,L-1)) + ABS(H(L,L))
         IF (S .EQ. 0.0) S = NORM
         IF (ABS(H(L,L-1)) .LE. MACHEP * S) GO TO 100
   80 CONTINUE
C     ********** FORM SHIFT **********
  100 X = H(EN,EN)
      IF (L .EQ. EN) GO TO 270
      Y = H(NA,NA)
      W = H(EN,NA) * H(NA,EN)
      IF (L .EQ. NA) GO TO 280
```

```
      IF (ITS .EQ. 30) GO TO 1000
      IF (ITS .NE. 10 .AND. ITS .NE. 20) GO TO 130
C     ********** FORM EXCEPTIONAL SHIFT **********
      T = T + X
C
      DO 120 I = LOW, EN
  120 H(I,I) = H(I,I) - X
C
      S = ABS(H(EN,NA)) + ABS(H(NA,ENM2))
      X = 0.75 * S
      Y = X
      W = -0.4375 * S * S
  130 ITS = ITS + 1
C     ********** LOOK FOR TWO CONSECUTIVE SMALL
C                SUB-DIAGONAL ELEMENTS.
C                FOR M=EN-2 STEP -1 UNTIL L DO -- **********
      DO 140 MM = L, ENM2
         M = ENM2 + L - MM
         ZZ = H(M,M)
         R = X - ZZ
         S = Y - ZZ
         P = (R * S - W) / H(M+1,M) + H(M,M+1)
         Q = H(M+1,M+1) - ZZ - R - S
         R = H(M+2,M+1)
         S = ABS(P) + ABS(Q) + ABS(R)
         P = P / S
         Q = Q / S
         R = R / S
         IF (M .EQ. L) GO TO 150
         IF (ABS(H(M,M-1)) * (ABS(Q) + ABS(R)) .LE. MACHEP * ABS(P)
     X    * (ABS(H(M-1,M-1)) + ABS(ZZ) + ABS(H(M+1,M+1)))) GO TO 150
  140 CONTINUE
C
  150 MP2 = M + 2
C
      DO 160 I = MP2, EN
         H(I,I-2) = 0.0
         IF (I .EQ. MP2) GO TO 160
         H(I,I-3) = 0.0
  160 CONTINUE
C     ********** DOUBLE QR STEP INVOLVING ROWS L TO EN AND
C                COLUMNS M TO EN **********
      DO 260 K = M, NA
         NOTLAS = K .NE. NA
         IF (K .EQ. M) GO TO 170
         P = H(K,K-1)
         Q = H(K+1,K-1)
         R = 0.0
         IF (NOTLAS) R = H(K+2,K-1)
         X = ABS(P) + ABS(Q) + ABS(R)
         IF (X .EQ. 0.0) GO TO 260
         P = P / X
         Q = Q / X
         R = R / X
  170    S = SIGN(SQRT(P*P+Q*Q+R*R),P)
         IF (K .EQ. M) GO TO 180
         H(K,K-1) = -S * X
         GO TO 190
```

```
  180    IF (L .NE. M) H(K,K-1) = -H(K,K-1)
  190    P = P + S
         X = P / S
         Y = Q / S
         ZZ = R / S
         Q = Q / P
         R = R / P
C     ********** ROW MODIFICATION **********
         DO 210 J = K, EN
            P = H(K,J) + Q * H(K+1,J)
            IF (.NOT. NOTLAS) GO TO 200
            P = P + R * H(K+2,J)
            H(K+2,J) = H(K+2,J) - P * ZZ
  200       H(K+1,J) = H(K+1,J) - P * Y
            H(K,J) = H(K,J) - P * X
  210    CONTINUE
C
         J = MIN0(EN,K+3)
C     ********** COLUMN MODIFICATION **********
         DO 230 I = L, J
            P = X * H(I,K) + Y * H(I,K+1)
            IF (.NOT. NOTLAS) GO TO 220
            P = P + ZZ * H(I,K+2)
            H(I,K+2) = H(I,K+2) - P * R
  220       H(I,K+1) = H(I,K+1) - P * Q
            H(I,K) = H(I,K) - P
  230    CONTINUE
C
  260 CONTINUE
C
      GO TO 70
C     ********** ONE ROOT FOUND **********
  270 WR(EN) = X + T
      WI(EN) = 0.0
      EN = NA
      GO TO 60
C     ********** TWO ROOTS FOUND **********
  280 P = (Y - X) / 2.0
      Q = P * P + W
      ZZ = SQRT(ABS(Q))
      X = X + T
      IF (Q .LT. 0.0) GO TO 320
C     ********** REAL PAIR **********
      ZZ = P + SIGN(ZZ,P)
      WR(NA) = X + ZZ
      WR(EN) = WR(NA)
      IF (ZZ .NE. 0.0) WR(EN) = X - W / ZZ
      WI(NA) = 0.0
      WI(EN) = 0.0
      GO TO 330
C     ********** COMPLEX PAIR **********
  320 WR(NA) = X + P
      WR(EN) = X + P
      WI(NA) = ZZ
      WI(EN) = -ZZ
  330 EN = ENM2
      GO TO 60
C     ********** SET ERROR -- NO CONVEZZ_RGENCE TO AN
```

```
C                 EIGENVALUE AFTER 30 ITERATIONS **********
 1000 IERR = EN
 1001 RETURN
C     ********** LAST CARD OF ZZ_HQR **********
      END
*DECK ZZ_ELTRAN
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_ELTRAN(NM,N,LOW,IGH,A,INT,Z)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NM,IGH),Z(NM,N)
      DIMENSION INT(IGH)
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ELMTRANS,
C     NUM. MATH. 16, 181-204(1970) BY PETERS AND WILKINSON.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 372-395(1971).
C
C     THIS SUBROUTINE ACCUMULATES THE STABILIZED ELEMENTARY
C     SIMILARITY TRANSFORMATIONS USED IN THE REDUCTION OF A
C     REAL GENERAL MATRIX TO UPPER HESSENBEZZ_RG FORM BY  ZZ_ELMHES.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
C
C        LOW AND IGH ARE INTEGERS DETERMINED BY THE ZZ_BALANCING
C          SUBROUTINE  ZZ_BALANC.  IF  ZZ_BALANC  HAS NOT BEEN USED,
C          SET LOW=1, IGH=N,
C
C        A CONTAINS THE MULTIPLIERS WHICH WERE USED IN THE
C          REDUCTION BY  ZZ_ELMHES  IN ITS LOWER TRIANGLE
C          BELOW THE SUBDIAGONAL,
C
C        INT CONTAINS INFORMATION ON THE ROWS AND COLUMNS
C          INTERCHANGED IN THE REDUCTION BY  ZZ_ELMHES.
C          ONLY ELEMENTS LOW THROUGH IGH ARE USED.
C
C     ON OUTPUT-
C
C        Z CONTAINS THE TRANSFORMATION MATRIX PRODUCED IN THE
C          REDUCTION BY  ZZ_ELMHES.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
C     ********** INITIALIZE Z TO IDENTITY MATRIX **********
      DO 80 I = 1, N
C
         DO 60 J = 1, N
   60    Z(I,J) = 0.0
```

```
C
         Z(I,I) = 1.0
   80 CONTINUE
C
      KL = IGH - LOW - 1
      IF (KL .LT. 1) GO TO 200
C     ********** FOR MP=IGH-1 STEP -1 UNTIL LOW+1 DO -- **********
      DO 140 MM = 1, KL
         MP = IGH - MM
         MP1 = MP + 1
C
         DO 100 I = MP1, IGH
  100    Z(I,MP) = A(I,MP-1)
C
         I = INT(MP)
         IF (I .EQ. MP) GO TO 140
C
         DO 130 J = MP, IGH
            Z(MP,J) = Z(I,J)
            Z(I,J) = 0.0
  130    CONTINUE
C
         Z(I,MP) = 1.0
  140 CONTINUE
C
  200 RETURN
C     ********** LAST CARD OF ZZ_ELTRAN **********
      END
*DECK ZZ_HQR2
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_HQR2(NM,N,LOW,IGH,H,WR,WI,Z,IERR)
C
C     Note: Intrinsic functions REAL, AIMAG, and CMPLX have been replaced
C     by their DOUBLE PRECISION equivalents DREAL, DIMAG, and DCMPLX.
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMPLEX*16 Z3
      INTEGER EN,ENM2
      DIMENSION H(NM,N),WR(N),WI(N),Z(NM,N)
      DOUBLE PRECISION NORM,MACHEP
      LOGICAL NOTLAS
C
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ZZ_HQR2,
C     NUM. MATH. 16, 181-204(1970) BY PETERS AND WILKINSON.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 372-395(1971).
C
C     THIS SUBROUTINE FINDS THE EIGENVALUES AND EIGENVECTORS
C     OF A REAL UPPER HESSENBEZZ_RG MATRIX BY THE QR METHOD.  THE
C     EIGENVECTORS OF A REAL GENERAL MATRIX CAN ALSO BE FOUND
C     IF  ZZ_ELMHES  AND  ZZ_ELTRAN  OR  ORTHES  AND  ORTRAN  HAVE
C     BEEN USED TO REDUCE THIS GENERAL MATRIX TO HESSENBEZZ_RG FORM
C     AND TO ACCUMULATE THE SIMILARITY TRANSFORMATIONS.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
```

```
C           ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C           DIMENSION STATEMENT,
C
C         N IS THE ORDER OF THE MATRIX,
C
C         LOW AND IGH ARE INTEGERS DETERMINED BY THE ZZ_BALANCING
C           SUBROUTINE  ZZ_BALANC.  IF  ZZ_BALANC  HAS NOT BEEN USED,
C           SET LOW=1, IGH=N,
C
C         H CONTAINS THE UPPER HESSENBEZZ_RG MATRIX,
C
C         Z CONTAINS THE TRANSFORMATION MATRIX PRODUCED BY  ZZ_ELTRAN
C           AFTER THE REDUCTION BY  ZZ_ELMHES, OR BY  ORTRAN  AFTER THE
C           REDUCTION BY  ORTHES, IF PERFORMED.  IF THE EIGENVECTORS
C           OF THE HESSENBEZZ_RG MATRIX ARE DESIRED, Z MUST CONTAIN THE
C           IDENTITY MATRIX.
C
C      ON OUTPUT-
C
C         H HAS BEEN DESTROYED,
C
C         WR AND WI CONTAIN THE REAL AND IMAGINARY PARTS,
C           RESPECTIVELY, OF THE EIGENVALUES.  THE EIGENVALUES
C           ARE UNORDERED EXCEPT THAT COMPLEX CONJUGATE PAIRS
C           OF VALUES APPEAR CONSECUTIVELY WITH THE EIGENVALUE
C           HAVING THE POSITIVE IMAGINARY PART FIRST.  IF AN
C           ERROR EXIT IS MADE, THE EIGENVALUES SHOULD BE CORRECT
C           FOR INDICES IERR+1,...,N,
C
C         Z CONTAINS THE REAL AND IMAGINARY PARTS OF THE EIGENVECTORS.
C           IF THE I-TH EIGENVALUE IS REAL, THE I-TH COLUMN OF Z
C           CONTAINS ITS EIGENVECTOR.  IF THE I-TH EIGENVALUE IS COMPLEX
C           WITH POSITIVE IMAGINARY PART, THE I-TH AND (I+1)-TH
C           COLUMNS OF Z CONTAIN THE REAL AND IMAGINARY PARTS OF ITS
C           EIGENVECTOR.  THE EIGENVECTORS ARE UNNORMALIZED.  IF AN
C           ERROR EXIT IS MADE, NONE OF THE EIGENVECTORS HAS BEEN FOUND,
C
C         IERR IS SET TO
C           ZERO       FOR NORMAL RETURN,
C           J          IF THE J-TH EIGENVALUE HAS NOT BEEN
C                      DETERMINED AFTER 30 ITERATIONS.
C
C      ARITHMETIC IS REAL EXCEPT FOR THE REPLACEMENT OF THE ALGOL
C      PROCEDURE CDIV BY COMPLEX DIVISION.
C
C      QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C      APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C      ------------------------------------------------------------------
C
C      ********** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                 THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
C
C                 **********
       MACHEP = 2.**(-47)
C
       IERR = 0
       NORM = 0.0
```

```
      K = 1
C     ********** STORE ROOTS ISOLATED BY ZZ_BALANC
C                AND COMPUTE MATRIX NORM **********
      DO 50 I = 1, N
C
         DO 40 J = K, N
   40    NORM = NORM + ABS(H(I,J))
C
         K = I
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 50
         WR(I) = H(I,I)
         WI(I) = 0.0
   50 CONTINUE
C
      EN = IGH
      T = 0.0
C     ********** SEARCH FOR NEXT EIGENVALUES **********
   60 IF (EN .LT. LOW) GO TO 340
      ITS = 0
      NA = EN - 1
      ENM2 = NA - 1
C     ********** LOOK FOR SINGLE SMALL SUB-DIAGONAL ELEMENT
C                FOR L=EN STEP -1 UNTIL LOW DO -- **********
   70 DO 80 LL = LOW, EN
         L = EN + LOW - LL
         IF (L .EQ. LOW) GO TO 100
         S = ABS(H(L-1,L-1)) + ABS(H(L,L))
         IF (S .EQ. 0.0) S = NORM
         IF (ABS(H(L,L-1)) .LE. MACHEP * S) GO TO 100
   80 CONTINUE
C     ********** FORM SHIFT **********
  100 X = H(EN,EN)
      IF (L .EQ. EN) GO TO 270
      Y = H(NA,NA)
      W = H(EN,NA) * H(NA,EN)
      IF (L .EQ. NA) GO TO 280
      IF (ITS .EQ. 30) GO TO 1000
      IF (ITS .NE. 10 .AND. ITS .NE. 20) GO TO 130
C     ********** FORM EXCEPTIONAL SHIFT **********
      T = T + X
C
      DO 120 I = LOW, EN
  120 H(I,I) = H(I,I) - X
C
      S = ABS(H(EN,NA)) + ABS(H(NA,ENM2))
      X = 0.75 * S
      Y = X
      W = -0.4375 * S * S
  130 ITS = ITS + 1
C     ********** LOOK FOR TWO CONSECUTIVE SMALL
C                SUB-DIAGONAL ELEMENTS.
C                FOR M=EN-2 STEP -1 UNTIL L DO -- **********
      DO 140 MM = L, ENM2
         M = ENM2 + L - MM
         ZZ = H(M,M)
         R = X - ZZ
         S = Y - ZZ
         P = (R * S - W) / H(M+1,M) + H(M,M+1)
```

```
         Q = H(M+1,M+1) - ZZ - R - S
         R = H(M+2,M+1)
         S = ABS(P) + ABS(Q) + ABS(R)
         P = P / S
         Q = Q / S
         R = R / S
         IF (M .EQ. L) GO TO 150
         IF (ABS(H(M,M-1)) * (ABS(Q) + ABS(R)) .LE. MACHEP * ABS(P)
     X    * (ABS(H(M-1,M-1)) + ABS(ZZ) + ABS(H(M+1,M+1)))) GO TO 150
  140 CONTINUE
C
  150 MP2 = M + 2
C
      DO 160 I = MP2, EN
         H(I,I-2) = 0.0
         IF (I .EQ. MP2) GO TO 160
         H(I,I-3) = 0.0
  160 CONTINUE
C     ********** DOUBLE QR STEP INVOLVING ROWS L TO EN AND
C                COLUMNS M TO EN **********
      DO 260 K = M, NA
         NOTLAS = K .NE. NA
         IF (K .EQ. M) GO TO 170
         P = H(K,K-1)
         Q = H(K+1,K-1)
         R = 0.0
         IF (NOTLAS) R = H(K+2,K-1)
         X = ABS(P) + ABS(Q) + ABS(R)
         IF (X .EQ. 0.0) GO TO 260
         P = P / X
         Q = Q / X
         R = R / X
  170    S = SIGN(SQRT(P*P+Q*Q+R*R),P)
         IF (K .EQ. M) GO TO 180
         H(K,K-1) = -S * X
         GO TO 190
  180    IF (L .NE. M) H(K,K-1) = -H(K,K-1)
  190    P = P + S
         X = P / S
         Y = Q / S
         ZZ = R / S
         Q = Q / P
         R = R / P
C     ********** ROW MODIFICATION **********
         DO 210 J = K, N
            P = H(K,J) + Q * H(K+1,J)
            IF (.NOT. NOTLAS) GO TO 200
            P = P + R * H(K+2,J)
            H(K+2,J) = H(K+2,J) - P * ZZ
  200       H(K+1,J) = H(K+1,J) - P * Y
            H(K,J) = H(K,J) - P * X
  210    CONTINUE
C
         J = MIN0(EN,K+3)
C     ********** COLUMN MODIFICATION **********
         DO 230 I = 1, J
            P = X * H(I,K) + Y * H(I,K+1)
            IF (.NOT. NOTLAS) GO TO 220
```

```
            P = P + ZZ * H(I,K+2)
            H(I,K+2) = H(I,K+2) - P * R
  220       H(I,K+1) = H(I,K+1) - P * Q
            H(I,K) = H(I,K) - P
  230    CONTINUE
C     ********** ACCUMULATE TRANSFORMATIONS **********
         DO 250 I = LOW, IGH
            P = X * Z(I,K) + Y * Z(I,K+1)
            IF (.NOT. NOTLAS) GO TO 240
            P = P + ZZ * Z(I,K+2)
            Z(I,K+2) = Z(I,K+2) - P * R
  240       Z(I,K+1) = Z(I,K+1) - P * Q
            Z(I,K) = Z(I,K) - P
  250    CONTINUE
C
  260 CONTINUE
C
      GO TO 70
C     ********** ONE ROOT FOUND **********
  270 H(EN,EN) = X + T
      WR(EN) = H(EN,EN)
      WI(EN) = 0.0
      EN = NA
      GO TO 60
C     ********** TWO ROOTS FOUND **********
  280 P = (Y - X) / 2.0
      Q = P * P + W
      ZZ = SQRT(ABS(Q))
      H(EN,EN) = X + T
      X = H(EN,EN)
      H(NA,NA) = Y + T
      IF (Q .LT. 0.0) GO TO 320
C     ********** REAL PAIR **********
      ZZ = P + SIGN(ZZ,P)
      WR(NA) = X + ZZ
      WR(EN) = WR(NA)
      IF (ZZ .NE. 0.0) WR(EN) = X - W / ZZ
      WI(NA) = 0.0
      WI(EN) = 0.0
      X = H(EN,NA)
      S = ABS(X) + ABS(ZZ)
      P = X / S
      Q = ZZ / S
      R = SQRT(P*P+Q*Q)
      P = P / R
      Q = Q / R
C     ********** ROW MODIFICATION **********
      DO 290 J = NA, N
         ZZ = H(NA,J)
         H(NA,J) = Q * ZZ + P * H(EN,J)
         H(EN,J) = Q * H(EN,J) - P * ZZ
  290 CONTINUE
C     ********** COLUMN MODIFICATION **********
      DO 300 I = 1, EN
         ZZ = H(I,NA)
         H(I,NA) = Q * ZZ + P * H(I,EN)
         H(I,EN) = Q * H(I,EN) - P * ZZ
  300 CONTINUE
```

```
C     ********** ACCUMULATE TRANSFORMATIONS **********
      DO 310 I = LOW, IGH
         ZZ = Z(I,NA)
         Z(I,NA) = Q * ZZ + P * Z(I,EN)
         Z(I,EN) = Q * Z(I,EN) - P * ZZ
  310 CONTINUE
C
      GO TO 330
C     ********** COMPLEX PAIR **********
  320 WR(NA) = X + P
      WR(EN) = X + P
      WI(NA) = ZZ
      WI(EN) = -ZZ
  330 EN = ENM2
      GO TO 60
C     ********** ALL ROOTS FOUND.  BACKSUBSTITUTE TO FIND
C                VECTORS OF UPPER TRIANGULAR FORM **********
  340 IF (NORM .EQ. 0.0) GO TO 1001
C     ********** FOR EN=N STEP -1 UNTIL 1 DO -- **********
      DO 800 NN = 1, N
         EN = N + 1 - NN
         P = WR(EN)
         Q = WI(EN)
         NA = EN - 1
         IF (Q) 710, 600, 800
C     ********** REAL VECTOR **********
  600    M = EN
         H(EN,EN) = 1.0
         IF (NA .EQ. 0) GO TO 800
C     ********** FOR I=EN-1 STEP -1 UNTIL 1 DO -- **********
         DO 700 II = 1, NA
            I = EN - II
            W = H(I,I) - P
            R = H(I,EN)
            IF (M .GT. NA) GO TO 620
C
            DO 610 J = M, NA
  610       R = R + H(I,J) * H(J,EN)
C
  620       IF (WI(I) .GE. 0.0) GO TO 630
            ZZ = W
            S = R
            GO TO 700
  630       M = I
            IF (WI(I) .NE. 0.0) GO TO 640
            T = W
            IF (W .EQ. 0.0) T = MACHEP * NORM
            H(I,EN) = -R / T
            GO TO 700
C     ********** SOLVE REAL EQUATIONS **********
  640       X = H(I,I+1)
            Y = H(I+1,I)
            Q = (WR(I) - P) * (WR(I) - P) + WI(I) * WI(I)
            T = (X * S - ZZ * R) / Q
            H(I,EN) = T
            IF (ABS(X) .LE. ABS(ZZ)) GO TO 650
            H(I+1,EN) = (-R - W * T) / X
            GO TO 700
```

```
  650       H(I+1,EN) = (-S - Y * T) / ZZ
  700    CONTINUE
C     ********** END REAL VECTOR **********
       GO TO 800
C     ********** COMPLEX VECTOR **********
  710    M = NA
C     ********** LAST VECTOR COMPONENT CHOSEN IMAGINARY SO THAT
C                EIGENVECTOR MATRIX IS TRIANGULAR **********
       IF (ABS(H(EN,NA)) .LE. ABS(H(NA,EN))) GO TO 720
       H(NA,NA) = Q / H(EN,NA)
       H(NA,EN) = -(H(EN,EN) - P) / H(EN,NA)
       GO TO 730
  720  Z3 = DCMPLX(0.0D0,-H(NA,EN)) / DCMPLX(H(NA,NA)-P,Q)
       H(NA,NA) = DREAL(Z3)
       H(NA,EN) = DIMAG(Z3)
  730  H(EN,NA) = 0.0
       H(EN,EN) = 1.0
       ENM2 = NA - 1
       IF (ENM2 .EQ. 0) GO TO 800
C     ********** FOR I=EN-2 STEP -1 UNTIL 1 DO -- **********
       DO 790 II = 1, ENM2
          I = NA - II
          W = H(I,I) - P
          RA = 0.0
          SA = H(I,EN)
C
          DO 760 J = M, NA
             RA = RA + H(I,J) * H(J,NA)
             SA = SA + H(I,J) * H(J,EN)
  760     CONTINUE
C
          IF (WI(I) .GE. 0.0) GO TO 770
          ZZ = W
          R = RA
          S = SA
          GO TO 790
  770     M = I
          IF (WI(I) .NE. 0.0) GO TO 780
          Z3 = DCMPLX(-RA,-SA) / DCMPLX(W,Q)
          H(I,NA) = DREAL(Z3)
          H(I,EN) = DIMAG(Z3)
          GO TO 790
C     ********** SOLVE COMPLEX EQUATIONS **********
  780     X = H(I,I+1)
          Y = H(I+1,I)
          VR = (WR(I) - P) * (WR(I) - P) + WI(I) * WI(I) - Q * Q
          VI = (WR(I) - P) * 2.0 * Q
          IF (VR .EQ. 0.0 .AND. VI .EQ. 0.0) VR = MACHEP * NORM
     X     * (ABS(W) + ABS(Q) + ABS(X) + ABS(Y) + ABS(ZZ))
          Z3 = DCMPLX(X*R-ZZ*RA+Q*SA,X*S-ZZ*SA-Q*RA) / DCMPLX(VR,VI)
          H(I,NA) = DREAL(Z3)
          H(I,EN) = DIMAG(Z3)
          IF (ABS(X) .LE. ABS(ZZ) + ABS(Q)) GO TO 785
          H(I+1,NA) = (-RA - W * H(I,NA) + Q * H(I,EN)) / X
          H(I+1,EN) = (-SA - W * H(I,EN) - Q * H(I,NA)) / X
          GO TO 790
  785     Z3 = DCMPLX(-R-Y*H(I,NA),-S-Y*H(I,EN)) / DCMPLX(ZZ,Q)
          H(I+1,NA) = DREAL(Z3)
```

```
            H(I+1,EN) = DIMAG(Z3)
  790    CONTINUE
C     ********** END COMPLEX VECTOR **********
  800 CONTINUE
C     ********** END BACK SUBSTITUTION.
C                VECTORS OF ISOLATED ROOTS **********
      DO 840 I = 1, N
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 840
C
         DO 820 J = I, N
  820    Z(I,J) = H(I,J)
C
  840 CONTINUE
C     ********** MULTIPLY BY TRANSFORMATION MATRIX TO GIVE
C                VECTORS OF ORIGINAL FULL MATRIX.
C                FOR J=N STEP -1 UNTIL LOW DO -- **********
      DO 880 JJ = LOW, N
         J = N + LOW - JJ
         M = MIN0(J,IGH)
C
         DO 880 I = LOW, IGH
            ZZ = 0.0
C
            DO 860 K = LOW, M
  860       ZZ = ZZ + Z(I,K) * H(K,J)
C
            Z(I,J) = ZZ
  880 CONTINUE
C
      GO TO 1001
C     ********** SET ERROR -- NO CONVEZZ_RGENCE TO AN
C                EIGENVALUE AFTER 30 ITERATIONS **********
 1000 IERR = EN
 1001 RETURN
C     ********** LAST CARD OF ZZ_HQR2 **********
      END
*DECK ZZ_BALBAK
C
C     ------------------------------------------------------------------
C
      SUBROUTINE ZZ_BALBAK(NM,N,LOW,IGH,SCALE,M,Z)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION SCALE(N),Z(NM,M)
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE ZZ_BALBAK,
C     NUM. MATH. 13, 293-304(1969) BY PARLETT AND REINSCH.
C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 315-326(1971).
C
C     THIS SUBROUTINE FORMS THE EIGENVECTORS OF A REAL GENERAL
C     MATRIX BY BACK TRANSFORMING THOSE OF THE CORRESPONDING
C     ZZ_BALANCED MATRIX DETERMINED BY  ZZ_BALANC.
C
C     ON INPUT-
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
```

```
C
C        N IS THE ORDER OF THE MATRIX,
C
C        LOW AND IGH ARE INTEGERS DETERMINED BY  ZZ_BALANC,
C
C        SCALE CONTAINS INFORMATION DETERMINING THE PERMUTATIONS
C          AND SCALING FACTORS USED BY  ZZ_BALANC,
C
C        M IS THE NUMBER OF COLUMNS OF Z TO BE BACK TRANSFORMED,
C
C        Z CONTAINS THE REAL AND IMAGINARY PARTS OF THE EIGEN-
C          VECTORS TO BE BACK TRANSFORMED IN ITS FIRST M COLUMNS.
C
C     ON OUTPUT-
C
C        Z CONTAINS THE REAL AND IMAGINARY PARTS OF THE
C          TRANSFORMED EIGENVECTORS IN ITS FIRST M COLUMNS.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO B. S. GARBOW,
C     APPLIED MATHEMATICS DIVISION, AZZ_RGONNE NATIONAL LABORATORY
C
C     ------------------------------------------------------------------
C
      IF (M .EQ. 0) GO TO 200
      IF (IGH .EQ. LOW) GO TO 120
C
      DO 110 I = LOW, IGH
         S = SCALE(I)
C     ********** LEFT HAND EIGENVECTORS ARE BACK TRANSFORMED
C                IF THE FOREGOING STATEMENT IS REPLACED BY
C                S=1.0/SCALE(I). **********
         DO 100 J = 1, M
  100    Z(I,J) = Z(I,J) * S
C
  110 CONTINUE
C     ********- FOR I=LOW-1 STEP -1 UNTIL 1,
C                IGH+1 STEP 1 UNTIL N DO -- **********
  120 DO 140 II = 1, N
         I = II
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 140
         IF (I .LT. LOW) I = LOW - II
         K = SCALE(I)
         IF (K .EQ. I) GO TO 140
C
         DO 130 J = 1, M
            S = Z(I,J)
            Z(I,J) = Z(K,J)
            Z(K,J) = S
  130    CONTINUE
C
  140 CONTINUE
C
  200 RETURN
C     ********** LAST CARD OF ZZ_BALBAK **********
      END
```